NEY YORK UNIVERSITY

CAPSTONE THESIS

Dynamic equivalency and peer grouping using Factor Modeling and Machine Learning

Author:
Jielan ZHENG, Yashu ZHU,
Jiangguangyu XUE

Supervisor:
Mr. Chris ROMANO

A thesis submitted in fulfillment of the requirements
for the degree of Master of Finance and Risk Engineering

August 13, 2019

NEY YORK UNIVERSITY

Abstract

Finance and Risk Department
Tandon School Of Engineering

Master of Finance and Risk Engineering

Dynamic equivalency and peer grouping using Factor Modeling and Machine Learning

by Jielan ZHENG, Yashu ZHU, Jiangguangyu XUE

This capstone project focus on cluster analysis of financial assets. Clustering is used as important knowledge discovery tools in modern machine learning process.The cluster analysis serves as a method to find which assets are different from each other. Given the wide spread use of factor modeling and machine learning, a firm should be able to dynamically create peer grouping. While diversification is a key step for constructing portfolios but true diversification is not possible as investing in each company requires a lot of capital. we discover the groupings of stocks which we get data from current stock market and transform data into organized structure through machine learning technics. With several clustering models and conduction of Well dividing stocks into groups with "similar characteristics" can help in portfolio construction to ensure we choose a universe of stocks with sufficient diversification between them...

Acknowledgements

We would like to extend our sincere gratitude to Mr. Chris Romano and Professor Agnes Tourin, for their expert guidance and constructive feedback throughout the course of this project. We appreciate the patience with which Mr.Chris would tackle our doubts and the time he would spare from his busy schedule to meet during the course of his work-day to discuss. His inputs to this project are invaluable. We would also like to thank the Finance and Risk Engineering department at NYU Tandon School of Engineering to give us the opportunity to work on this advanced topic for our capstone project, during the course of which we learned a lot. ...

Team August 13, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The goal of this project is to conduct peer grouping of SP500 components using machine learning-based algorithms. This capstone project does the following several things coded in Python: It first identify and acquire the technical and fundamental data acquisition of SP500 components; built python framework for clustering; then conduct feature engineering to improve dataset by selecting representatively technical indicators; thirdly utilize PCA to perform dimension reduction on fundamental and technical features, then remove outlier dataset and built the final database; main part including conducting several clustering models with train and test sample from the database, then apply flexible evaluation approaches to relative models and analysis the clustering performance under each model. The program has to run in order as described.

The whole python framework design is shown in two main parts: Data and Clustering Model application. The data part is shown in the flow chart Figure 1.1.

In this project, the grouping stocks pool is SP500 components. To get the best results from the models and algorithms, it first involves getting the most out of the data for your algorithms to work with and so the data. This is the process and practice of feature engineering solves. Its main purposes are 1). Extract and reduce the high noise contained in the raw input. 2). To generate features containing sufficient and useful information needed for stock clustering. 3). Combine both technical features and fundamental features universally used in stock models.

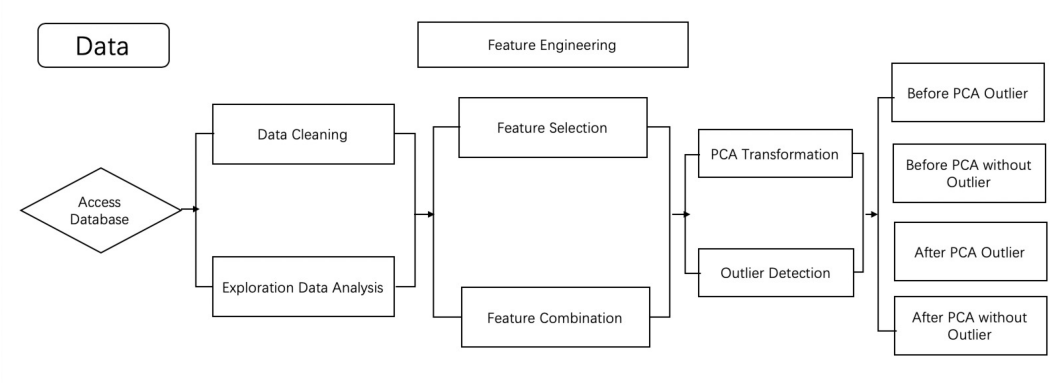

FIGURE 1.1: Dataframe Construction

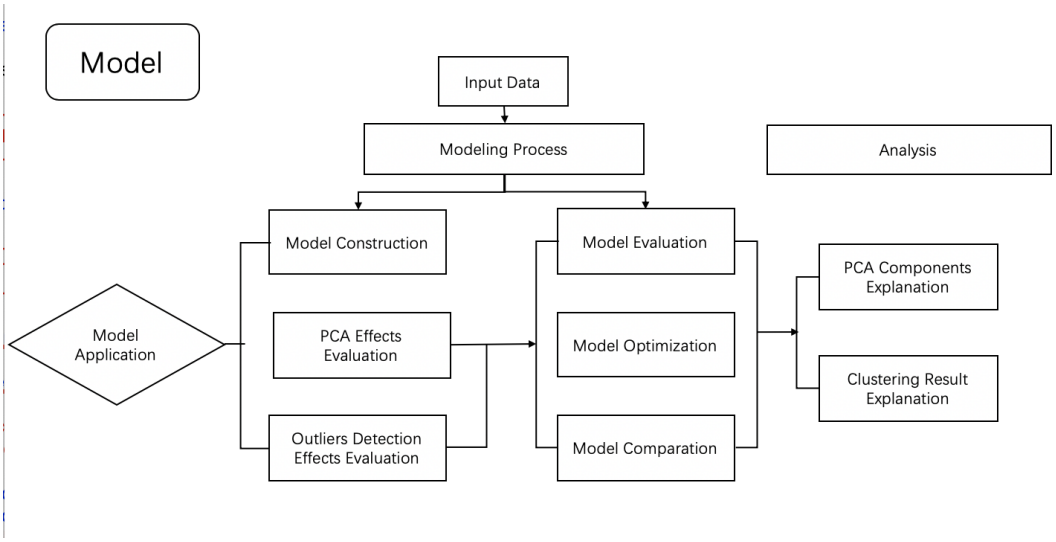

FIGURE 1.2: Model Construction

Moreover, we involved dimensionality reduction technique, Principal components analysis (PCA) and also remove outlier to achieve new database that 1). reduce dimensional feature-space 2). maintain as much information from the original dataset as possible in the reduced dataset. 3). exclude outlier noisy to the main sample.

The data part is shown in the flow chart Figure 1.2.

In the modeling process, we research and conduct several clustering models. To test the efficiency of PCA and then apply optimal datasets after PCA and exculpation of outliers as input to clustering models including K-means, MiniBatchKmeans, BIRCH, Agglomerative, Spectral, DBSCAN, Mean-Shift. To conduct properly valuation of performance under different clustering models, we work on several evaluation methods: 1). Calinski-Harabasz score 2). Silhouette Coefficient 3). Elbow Method 4). Gap Statistic Method.

Additionally, we manually assign the outliers into the "most similar'clustering group based on the distance calculation. Then compare model performances with full-dataset and dataset with outliers to test the distance metrics in grouping outliers and optimize final model results. Last, review and analysis all the model performances, draw the explanation of both PCA components analysis and peer grouping results.

We proceed as follows. In Chapter 2: Background, we provide some backgrounds in feature engineering, peer grouping model and machine learning techniques we used. Chapter 3: Data, we describe how we retrieve, parse and built data frame input. In Chapter 4: Clustering Model, we describe our modeling process, how we train, build and optimize our models, and relative evaluation implement included. In Chapter 5: Analysis we provide details of model performance and give an explanation of grouping results, and PCA factors analysis. In Chapter 6: Conclusion, we reach the conclusion based on all the analysis and reach and leave confusion and assumptions for further research. Last appendix where source codes will be provided.

Chapter 2

Background

2.1 Feature Engineering

Feature engineering is a term of art for data science and machine learning which refers to pre-processing and transforming raw data into a form which is more easily used by machine learning algorithms. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive. The need for manual feature engineering can be obviated by automated feature learning. This topic covers techniques for extracting and transforming features-the numeric representations of raw data-into formats for machine-learning models.[1]

Much like industrial processing can extract pure gold from trace elements within raw ore, feature engineering can extract valuable "alpha" from very noisy raw data.

2.1.1 Feature Engineering in stock data

Much research has been done in this field but it has been difficult to find the right features, which will help predict and classify with high accuracy. When you try to analyze stock classification using historical stock data, it has many attributes (feature set) containing useful as well as redundant features. Thus there is a need to remove the unwanted stock information.[2] In this research we first try to find the best features from the available company data as well as data about similar companies and stock indexes. The best-extracted features are then used to classify stock using different machine learning algorithms. [3]

2.2 Principle Component Analysis

Principle Component Analysis (PCA) is used for reducing the number of variables comprising a dataset while retaining the variability in the data, or identifying hidden patterns in the data, and classifying them according to how much of the information, stored in the data, they account for[4]. We use PCA for our stock market data for both purposes.

We take stock price panel data as score matrix A, and coefficient vector l that generates linear combination on A which yields principle components Y of smaller dimension than A. Principle components Y are independent and each coefficient vector li is required to maximize the variance of its corresponding principle component Y_i , as shown:

$$\max Var(Y_i) = l_i^T C l_i \tag{2.1}$$

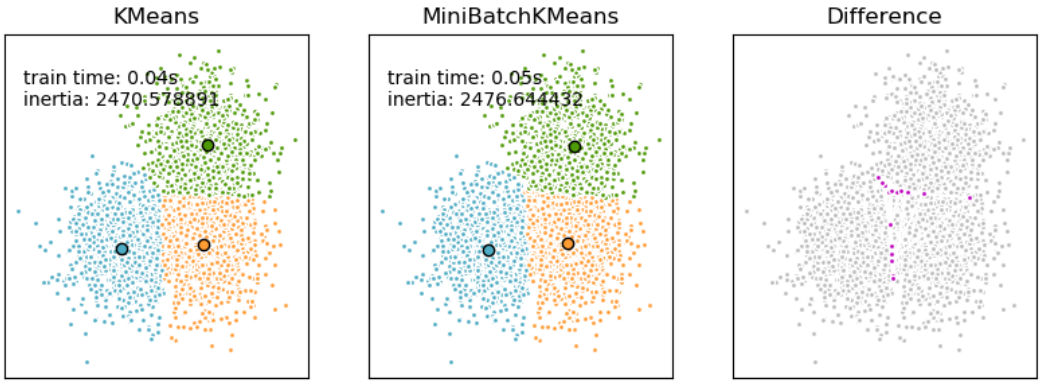

FIGURE 2.1: KMeans and MiniBatchKmeans

which has Lagrangian form:

$$s.t \qquad \|l_i^T\| = 1, \forall i$$
$$l_i^T l_j = 0, \forall i \neq j$$

Maximizing L by taking partial derivatives to 0, we obtain $Cl_i = \lambda_i l_i$, where C is covariance matrix of A and λ_i, $l_i$are corresponding eigenvalues and eigenvectors. Therefore, principle components are constructed from eigenvectors of score matrix with score matrix itself: $Y_i = l_i^T A$.

2.3 Clustering[5]

2.3.1 K-means and MiniBatchKmeans

K-means is often referred to as Lloyd's algorithm. In basic terms, the algorithm has three steps. The first step chooses the initial centroids, with the most basic method being to choose k samples from the dataset X. After initialization, K-means consists of looping between the two other steps. The first step assigns each sample to its nearest centroid. The second step creates new centroids by taking the mean value of all of the samples assigned to each previous centroid. The difference between the old and the new centroids are computed and the algorithm repeats these last two steps until this value is less than a threshold. In other words, it repeats until the centroids do not move significantly

The MiniBatchKMeans a variant of the KMeans algorithm which uses mini-batches to reduce the computation time, while still attempting to optimise the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence time of k-means, mini-batch k-means produces results that are generally only slightly worse than the standard algorithm.

The algorithm iterates between two major steps, similar to vanilla k-means. In the first step, b samples are drawn randomly from the dataset, to form a mini-batch. These are then assigned to the nearest centroid. In the second step, the centroids are updated. In contrast to k-means, this is done on a per-sample basis. For each sample in the mini-batch, the assigned centroid is updated by taking the streaming average

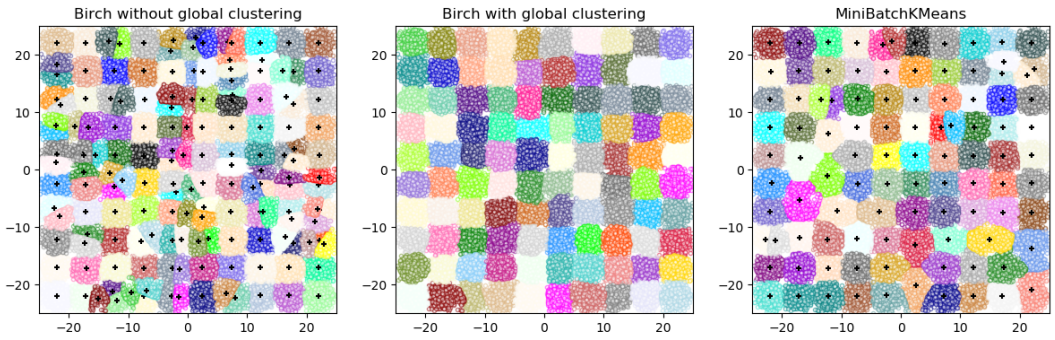

FIGURE 2.2: Birch

of the sample and all previous samples assigned to that centroid. This has the effect of decreasing the rate of change for a centroid over time. These steps are performed until convergence or a predetermined number of iterations is reached.

2.3.2 Birch

The Birch builds a tree called the Characteristic Feature Tree (CFT) for the given data. The data is essentially lossy compressed to a set of Characteristic Feature nodes (CF Nodes). The CF Nodes have a number of subclusters called Characteristic Feature subclusters (CF Subclusters) and these CF Subclusters located in the non-terminal CF Nodes can have CF Nodes as children. A new sample is inserted into the root of the CF Tree which is a CF Node. It is then merged with the subcluster of the root, that has the smallest radius after merging, constrained by the threshold and branching factor conditions. If the subcluster has any child node, then this is done repeatedly till it reaches a leaf. After finding the nearest subcluster in the leaf, the properties of this subcluster and the parent subclusters are recursively updated. If this split node has a parent subcluster and there is room for a new subcluster, then the parent is split into two. If there is no room, then this node is again split into two and the process is continued recursively, till it reaches the root. Birch does not scale very well to high dimensional data. As a rule of thumb if the number of feature is greater than twenty, it is generally better to use MiniBatchKMeans.

2.3.3 Hierarchical clustering

Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree (or dendrogram). The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample.

The AgglomerativeClustering object performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together.

AgglomerativeClustering can also scale to large number of samples when it is used jointly with a connectivity matrix, but is computationally expensive when no connectivity constraints are added between samples: it considers at each step all the possible merges.

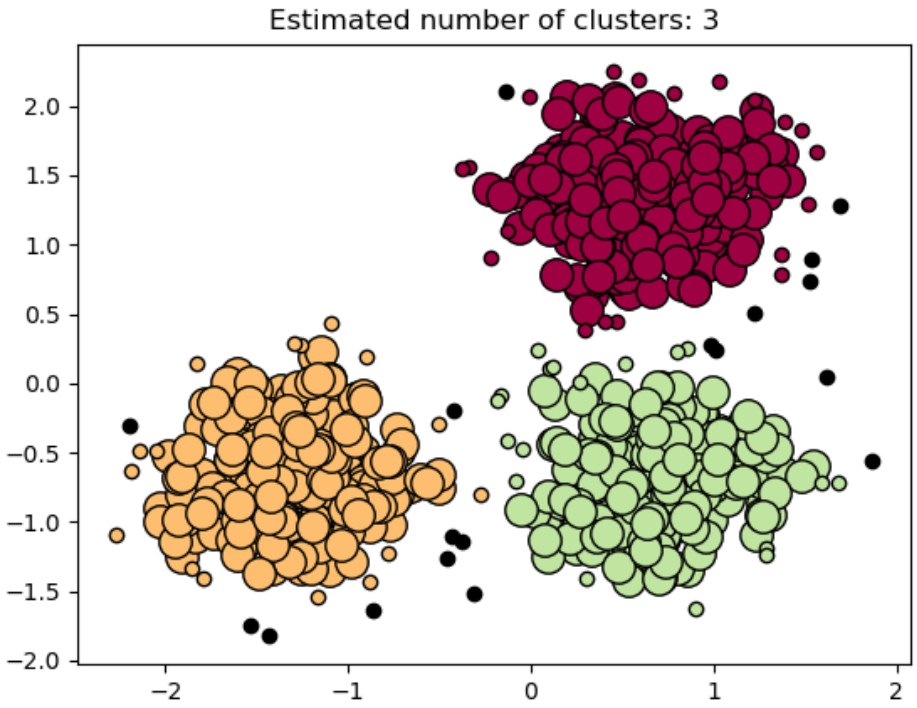

FIGURE 2.3: DBSCAN

2.3.4 SpectralClustering

SpectralClustering does a low-dimension embedding of the affinity matrix between samples, followed by a KMeans in the low dimensional space. It is especially efficient if the affinity matrix is sparse and the pyamgmodule is installed. SpectralClustering requires the number of clusters to be specified. It works well for a small number of clusters but is not advised when using many clusters.

2.3.5 DBSCAN

The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to k-means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples).

2.3.6 Mean Shift

MeanShift clustering aims to discover blobs in a smooth density of samples. It is a centroid based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate near-duplicates to form the final set of centroids.

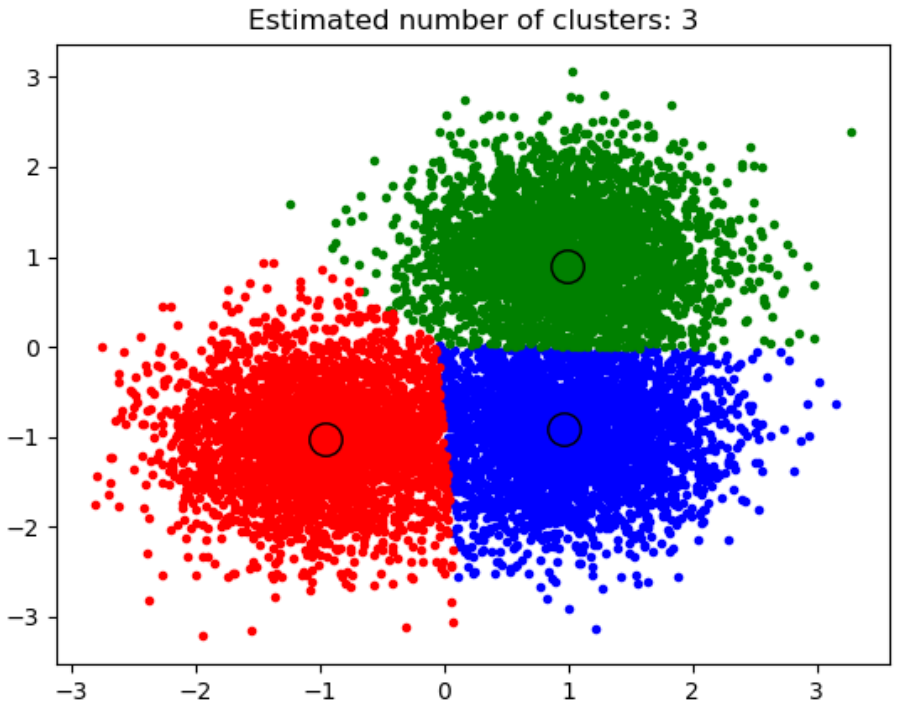

FIGURE 2.4: Meanshift

The algorithm is not highly scalable, as it requires multiple nearest neighbor searches during the execution of the algorithm. The algorithm is guaranteed to converge, however the algorithm will stop iterating when the change in centroids is small.

2.3.7 Affinity Propagation

Affinity Propagation creates clusters by sending messages between pairs of samples until convergence. A dataset is then described using a small number of exemplars, which are identified as those most representative of other samples. The messages sent between pairs represent the suitability for one sample to be the exemplar of the other, which is updated in response to the values from other pairs. This updating happens iteratively until convergence, at which point the final exemplars are chosen, and hence the final clustering is given.

2.4 Distance Metrics

Clustering is a technique to categorize the data into groups. Distance metrics plays a very important role in the clustering process. The more the similarity among the data in clusters, more the chances of particular data-items to belong to particular group. There are number of algorithms which are available for clustering.[6]

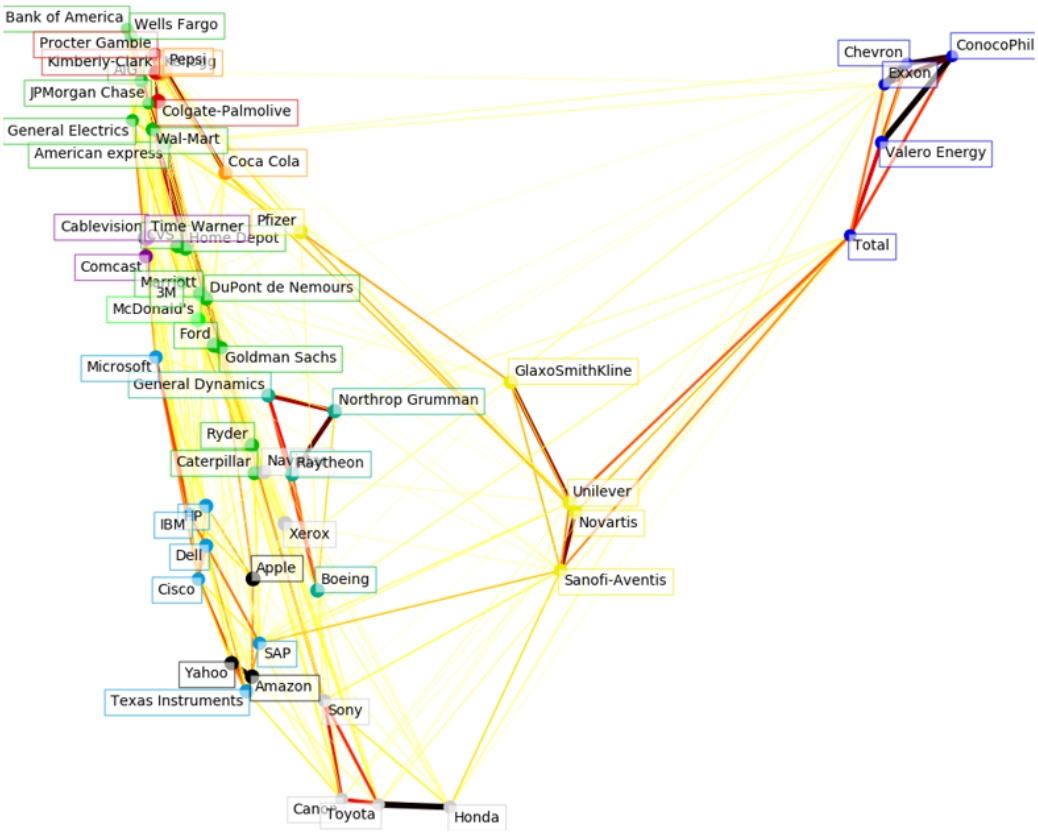

FIGURE 2.5: Affinity Propagation Example

2.4.1 Euclidean Distance

Euclidean distance computes the root of square difference between co-ordinates of pair of objects.

$$Dist_{XY} = \sqrt{\sum_{k=1}(X_{ik} - X_{jk})^2} \tag{2.2}$$

2.4.2 Manhattan Distance

Manhattan distance computes the absolute differences between coordinates of pair of objects

$$Dist_{XY} = |X_{ik} - X_{jk}| \tag{2.3}$$

2.4.3 Chebychev Distance

Chebychev Distance is also known as maximum value distance and is computed as the absolute magnitude of the differences between coordinate of a pair of objects

$$Dist_{XY} = \max |X_{ik} - X_{jk}| \tag{2.4}$$

2.4.4 Minkowski Distance

Minkowski Distance is the generalized metric distance

$$Dist_{XY} = \left[\sum_{k=1}^{d} |X_{ik} - X_{jk}|^{1/p}\right]^p \tag{2.5}$$

Note that when p=2, the distance becomes the Euclidean distance. When p=1 it becomes city block distance. Chebyshev distance is a variant of Minkowski distance where p= (taking a limit). This distance can be used for both ordinal and quantitative variables.

2.5 Clustering performance evaluation

Evaluating the performance of a clustering algorithm is not as trivial as counting the number of errors or the precision and recall of a supervised classification algorithm. In particular any evaluation metric should not take the absolute values of the cluster labels into account but rather if this clustering define separations of the data similar to some ground truth set of classes or satisfying some assumption such that members belong to the same class are more similar that members of different classes according to some similarity metric.[7]

2.5.1 Calinski-Harabasz score

If the ground truth labels are not known, the Calinski-Harabasz index can be used to evaluate the model, where a higher Calinski-Harabasz score relates to a model with better defined clusters. For k clusters, the Calinski-Harabasz score s is given as the ratio of the between-clusters dispersion mean and the within-cluster dispersion:

$$s(k) = \frac{T_r(B_k)}{T_r(W_k)} \times \frac{N-k}{k-1}$$

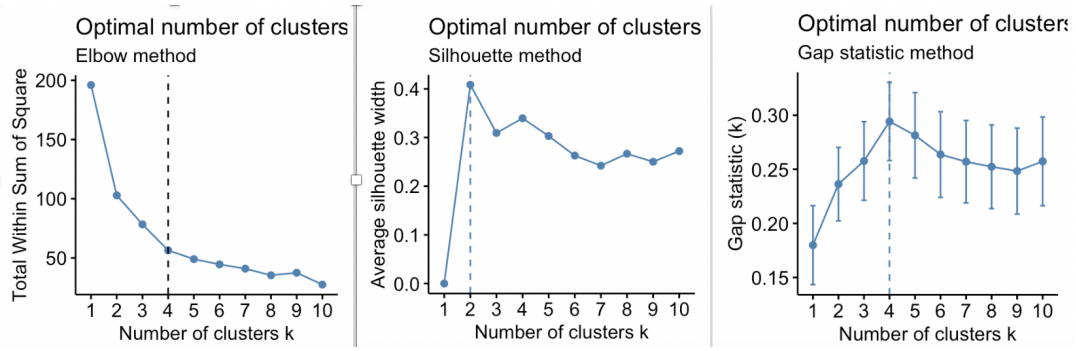

FIGURE 2.6: Elbow

where B_k is the between group dispersion matrix and W_k is the within-cluster dispersion matrix defined by:

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T$$
$$B_k = \sum_{q} n_q (c_q - c)(c_q - c)^T$$

with N be the number of points in our data, C_q be the set of points in cluster q, $c_q$be the center of cluster q, c be the center of E, n_q be the number of points in cluster q.

2.5.2 Silhouette Coefficient score

If the ground truth labels are not known, evaluation must be performed using the model itself. Higher Silhouette Coefficient score relates to a model with better defined clusters. The Silhouette Coefficient is defined for each sample and is composed of two scores:
• a: The mean distance between a sample and all other points in the same class.
• b: The mean distance between a sample and all other points in the next nearest cluster. The Silhouette Coefficient s for a single sample is then given as:

$$s = \frac{b - a}{max(a, b)}$$

2.5.3 Elbow method

The total WSS measures the compactness of the clustering and we want it to be as small as possible. The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

2.5.4 Gap statistic

The gap statistic compares the total within intra-cluster variation for different values of k with their expected values under null reference distribution of the data.

The estimate of the optimal clusters will be value that maximize the gap statistic (i.e, that yields the largest gap statistic). This means that the clustering structure is far away from the random uniform distribution of points.[8]

Chapter 3

Data

We choose a dataset of stocks listed in Standard and Poor 500. The choice is made considering S&P 500 will provide our dataset proper diversity since the stocks span a variety of industries. According to our study, a well-diversified dataset is more suitable for clustering purpose [9]. The next step is to determine the appropriate features to use for implementation. Different financial datasets have been considered. One common choice for input is technical indicators of each stock. For instance, Wittman's analysis was based on three-year historical price data of 91 stocks [10]. Another team utilized the daily prices of S&P 500 stocks within a trading year as the input of their model [11]. Both the experiment generated reasonable results. Apart from technical features, fundamental data from financial statements are important data source for clustering analysis as well. Since all public companies are required to file financial reports periodically, fundamental indicators found in those reports should have relatively small noise and trustworthy data records [12]. In the experiment of Momeni et al., five fundamental ratios (ROA, ROE, Profit to sales ratio, EPS, and Operating profit margins) are selected for their K-means model [13]. Hence, both fundamental and technical features are considered for our clustering analysis.

3.1 Data Collection

All the quantitative data are collected from Yahoo Finance using a python library yahoofinancials. To get data from Yahoo Finance, we first get the stock tickers from Wikipedia. The technical part consists of daily prices (adjusted close, high, low) and volume of each S&P 500 listed stocks from Jun 2018 to Jun 2019. Technical data for S&P 500 index of the same time period is also retrieved for further feature engineering purpose. The fundamental data contains 51 key statistics data along with earnings per share, market cap, PE ratio, net income, and EBIT of each stocks. They will be further processed to generate the features we need for our model.

3.2 Feature Engineering

3.2.1 Technical Features

For technical data, there are three stocks (CTVA, DOW, ZTS) that have unmatching date variables with the others (the median of the group). We regard those as unqualified entries since it may cause errors when transforming them to pandas dataframe. Hence, those stock entries are dropped from the dataset. By researching relative case studies, considering both risk and return prospect, we choose the standard deviation of daily return and volume, the skewness and kurtosis of daily

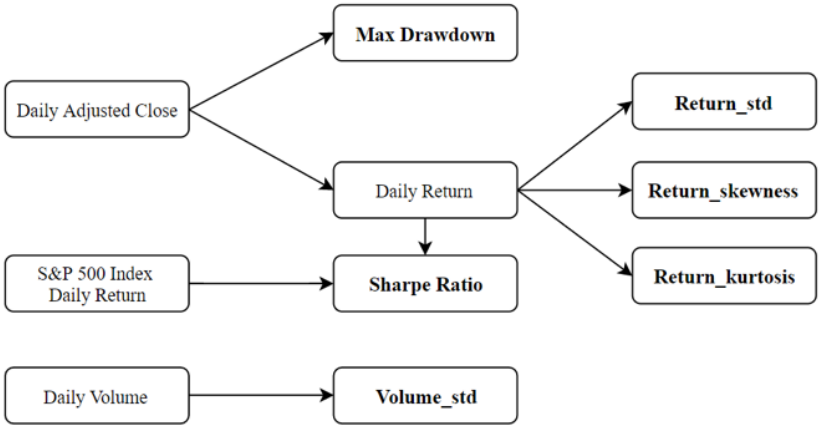

FIGURE 3.1: Feature Engineering for Technical Data

ticker	Return_std	Return_skew	Return_kurtosis	Sharpe_Ratio	Volume_std	Maxdrawdown
A	0.043125	0.572325	6.139398	0.029342	1.295404e+06	0.184895
AAL	0.059415	1.234017	8.127841	0.019145	3.349883e+06	0.386307
AAP	0.037719	-0.349844	9.141565	0.013477	5.052240e+05	0.305764
AAPL	0.048445	-0.566217	9.829872	0.015749	1.402302e+07	0.385159
ABBV	0.034534	0.708604	9.355761	-0.001167	2.907568e+06	0.206139

FIGURE 3.2: Processed Technical Features

return, Sharpe Ratio, and max drawdown as our technical features. Then, as Figure 3.1 suggested, they are calculated based on our processed technical data frame. These six factors are our technical input for clustering analysis (Figure 3.2).

3.2.2 Fundamental Features

For fundamental data, 57 indicators are collected originally. Regarding our existing dataset, here are two types of: (1) full of null entries (2) full of duplicated entries. Since no information is provided in those indicators, they are dropped out from our dataset. Then, to avoid comparison error in later steps, infinity entries are replaced by null value, which will be impute with mean or mode based on the pipeline setting later. After a series of processes, as Table 3.1 suggested, there are 28 fundamental features selected for model implementation.

3.2.3 Feature Combination

In this step, we combine the technical features and fundamental features into one single data frame for our clustering analysis. Due to the unmatching date (technical part) and the availability of financial statements (fundamental part) problem, two data frames are combined using inner join method. The integrated data frame is

TABLE 3.1: Fundamental Features and Explanation

Variables	Description
enterpriseToRevenue	Enterprise Value/Revenue, compares a company's enterprise value to its revenue
profitMargins	An indicator of a company's pricing strategies and how well it controls costs
enterpriseToEbitda	Enterprise Value/EBITDA, a popular valuation multiple used in the finance industry to measure the value of a company
52WeekChange	Movement of share prices in 52 weeks time frame
forwardEps	Earning per share calculated using Future earnings figures
sharesOutstanding	A company's stock currently held by all its shareholders
bookValue	The value of a security or asset as entered in a company's books
sharesShort	Amount of shares short selling
sharesPercentSharesOut	Short of Shares Outstanding
heldPercentInstitutions	Held by Institutions
netIncomeToCommon	Amount of net income to common shares
trailingEps	A company's earnings per sharegenerated over a prior period
SandP52WeekChange	S&P500 52-Week Change
priceToBook	Price to Book ratio
heldPercentInsiders	Held by Insiders
shortRatio	Calculated amount of short interest that exists in a stock
floatShares	Shares of a public corporation that are available for trading in a stockmarket
Beta	Beta (3Y Monthly), a measure of volatility or systematic risk
enterpriseValue	A measure of a company's total value
earningsQuarterlyGrowth	An increase of a company's sales when compared to aprevious quarter's revenue performance
pegRatio	A stock's price-to-earnings(P/E) ratio divided by the growth rate of its earnings for a specified time period
forwardPE	Calculated by as current stock price over thepredicted next annual earnings period
shortPercentOfFloat	Short of Float
sharesShortPriorMonth	Shares Short (prior month)
eps	Portion of a company's profit allocated to each outstanding share of common stock
Market Cap	The value or capitalization the market puts on a company
PE	The ratio for valuing a company that measure its current share price relative to its per-share earnings
NI	The difference after factoring deductions and taxes into gross income
EBIT	Earnings before interest and tax

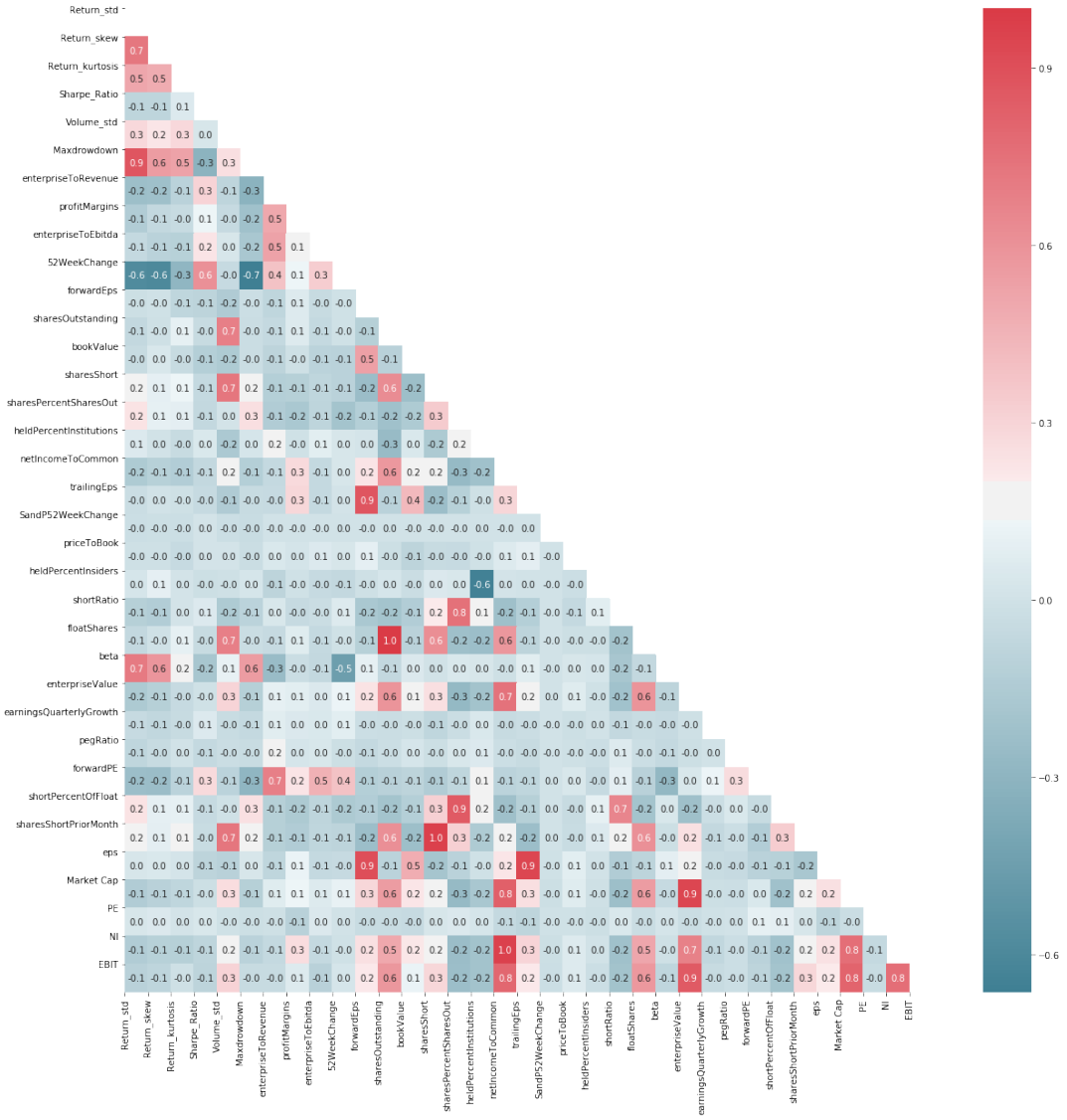

FIGURE 3.3: Correlations Heat map

then fed into our pipeline to impute missing values with corresponding mean and normalize all the entries. After such transformation, the data frame is qualified for dimensionality reduction.

3.3 Principal Component Analysis (PCA)

PCA is a common technique to reduce dimensions of features before training models. Since there are strong correlations among fundamental features, applying PCA here can help reduce the dimensions to a smaller number. And this can assist us in the future training of the clustering models because some of the models might work better with fewer dimensions.

As Figure 3.3 suggests, there are certain correlations among features we picked. This confirms our idea that there are strong correlations between fundamental features and PCA can be helpful here to reduce dimensions.

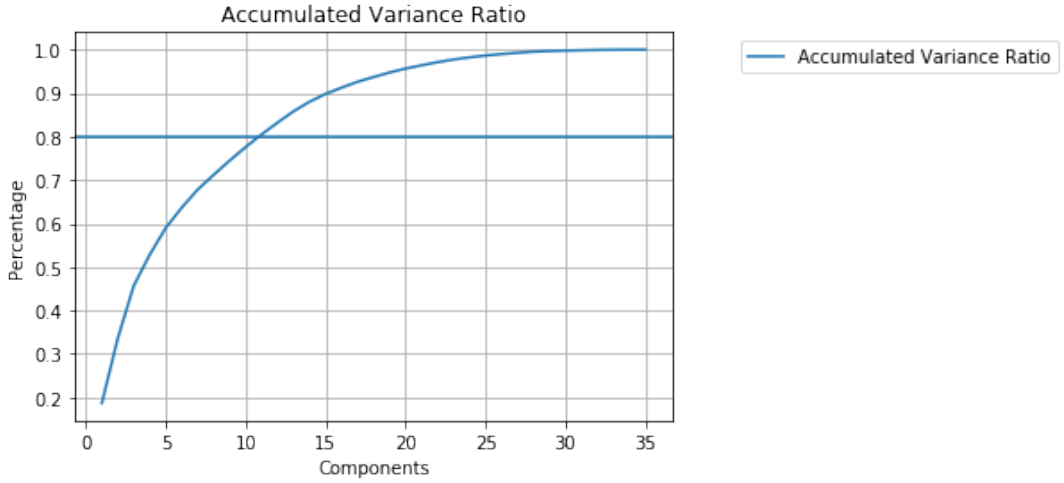

FIGURE 3.4: Cumulative Variance of Componets

Then, we utilized PCA function in Scikit-Learn package to do the analysis and transformation of features.

To determine the number of dimensions we need to keep after transformation, the indicator of cumulative variance explained is focused in next stage. The components are sorted according to the cumulative variance explained and shown in the Figure 3.4.

As shown in Figure 3.4, if we keep top 11 components, we will have over 80 % variance explained. In other words, instead of using 35 dimensions to refer to 100 % information, we can just use 11 components to refer to 80 % of the information. That is efficient and satisfying.

However, we transform real world indicators (e.g. EPS, Return, PE) into 11 components with unknown meanings. To explore and get better understanding of the transformed components, we decide to dive into the loadings matrix to see the connections between components and original features.

To do that, for each component, in the loading matrix, we filter the features contribute over 0.3 or below -0.3. And we name the 11 components according to those original features that contribute more to the components. For example, according to loadings matrix,

$$\text{Component 2} = \text{Return \_ std} \times 0.32 + \text{Maxdrowdown} \times 0.33 + \text{Others}$$

There are only two features that have weights over 0.3 and these two features are related to risk. As a result, we can name component 2 as Risk Factor.

Similarly, we can name all the components according to loadings matrix (Figure 3.5).

10 of the 11 components can be given suitable name, while 8th component is hard to explain. The meaning is not explicit and we will just call it factor 8.

Generally speaking, after PCA transformation, we have factors related to Company Scale, Risk, EPS, Shorting ratio, Overall value to profitability, Holding Positions, Standard Deviation and so on.

Component	Name	Main constitutions		Component	Name	Main constitutions	
1	Company Scale Factor	sharesOutstanding	0.32	6	Insiders Shares percents	heldPercentInsiders	0.67
		netIncomeToCommon	0.35			heldPercentInstitutions	-0.59
		floatShares	0.32	7	Standard diviation of Volume	Volume_std	0.30
		enterpriseValue	0.34	8	Factor 8	pegRatio	0.54
		Market Cap	0.35			Sharpe_Ratio	-0.33
		NI	0.33			priceToBook	-0.32
		EBIT	0.34			PE	-0.30
2	Risk Factor	Return_std	0.32	9	Price to Profitability Ratio(The larger the worse)	pegRatio	0.31
		Maxdrowdown	0.33			PE	0.61
3	EPS Factor	forwardEps	0.36			profitMargins	-0.53
		trailingEps	0.35	10	Price to Book Ratio	priceToBook	0.77
		eps	0.36			Sharpe_Ratio	-0.30
4	Short Factor	sharesPercentSharesOut	0.44	11	Quarter Earnings Growth	earningsQuarterlyGrowth	0.94
		shortRatio	0.45				
		shortPercentOfFloat	0.43				
5	Oerval value to Profitability (the larger the worse)	enterpriseToRevenue	0.41				
		enterpriseToEbitda	0.35				
		forwardPE	0.35				

FIGURE 3.5: Name of Componets

3.4 Outlier Detection

Clustering method can be sensitive to outliers. A few outliers might cause clustering models to create unnecessary clusters. Consequently, dealing with outliers before fitting the models can be crucial.

We implemented outliers' detections and removals separately to features after PCA and features before PCA. We use 5-sigma rule to judge whether a sample is outlier: for a sample, if any of its feature is out of range of $[\mu - 5\sigma, \mu + 5\sigma]$, we classify it into outliers.

After that, we have four datasets: dataset after PCA with outliers, dataset after PCA without outliers, dataset before PCA with outliers and dataset before PCA without outliers.

These four datasets are used in different clusters. We compared the results to see whether PCA transformations and outliers detections improve models performance.

Chapter 4

Clustering Model

4.1 KMeans

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

And there are two main parameters we should assign before we fit the model. Set algorithm = full, which represents the classical EM-style algorithm and supports sparse data. The n_clusters presents the clustering number.

4.1.1 PCA Effect Evaluation

Test if PCA improve data structure in clustering model based on silhouette score(SH score) as evaluation method. Use dataset before PCA and after PCA, and modeling with optimal parameters. As presented in the graph below. The location of the maximum is considered as the appropriate number of clusters.

The evaluation result in the Figure 4.1 shows PCA improve the data structure and dataset after PCA has general higher SH score than that before PCA.

We continue the research dataset with after PCA. The optimal clustering number of After PCA is 3 and SH score is 0.207.

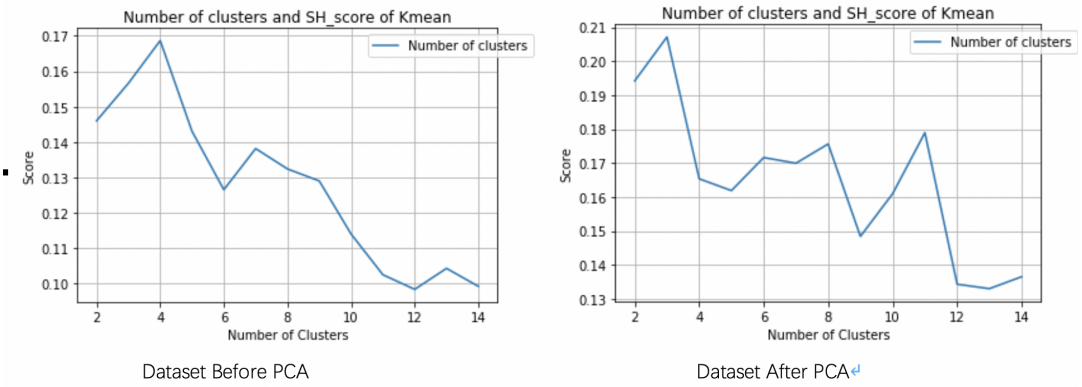

FIGURE 4.1: PCA Effect Evaluation

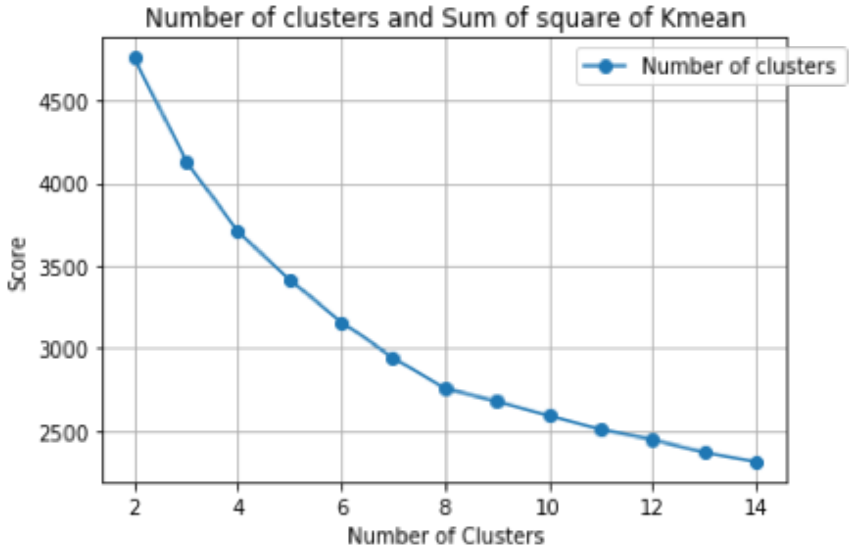

FIGURE 4.2: Elbow Method Of KMeans

4.1.2 Elbow Method

As mentioned in the background part, the locopation of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters. The curve in Figure 4.2 is not very explicit shows the optimal cluster number, and it only give a implicit range from 4 to 8.

4.1.3 Gap Statistic Method

The third evaluation method shows the results of the calculated gaps, the higher the value, the more optimal it is. We can see the relative optimal clusters number is 14 which is great different from the result above. But we could use the gap value of 3, 4, 5, 6, 8 as additional reference to evaluate other two metrics above. (Figure 4.3)

4.1.4 Prediction Distribution

Based on Figure 4.3 we see 8 and 11 as relative optimal clustering numbers. And the distribution graphs (Figure 4.4) show the prediction distribution of 8 and 11 clusters are both balanced.

4.1.5 Outliers Assignment

We remove outlier and find the relative clustering numbers above. Next we reassign those outlier based on the distance between each outlier and each cluster center. In the code part you can see the distance metric we use. After assignment, the SH score decrease as we expected and we do the assignment analyst in Chapter4.

4.2 MiniBatchKMeans

The MiniBatchKMeans is a variant of the KMeans algorithm which uses mini-batches to reduce the computation time, while still attempting to optimise the same

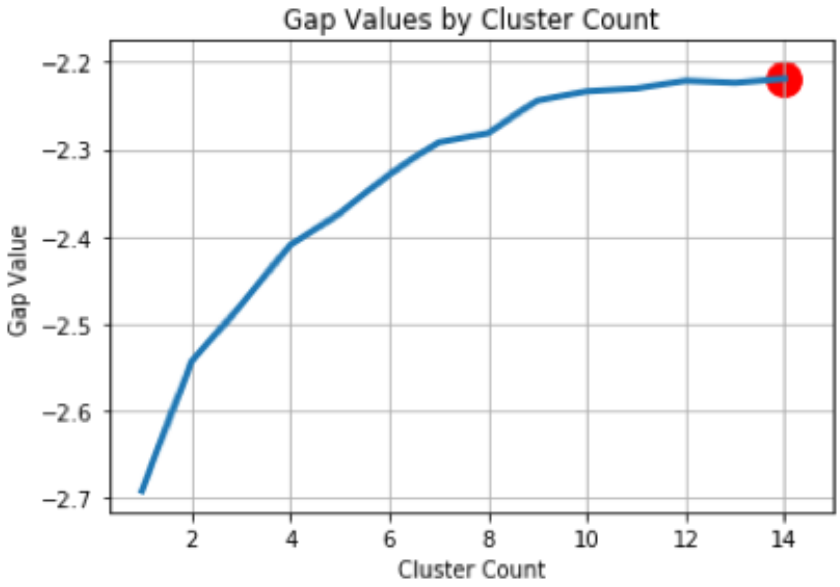

FIGURE 4.3: Gap Statistic KMeans

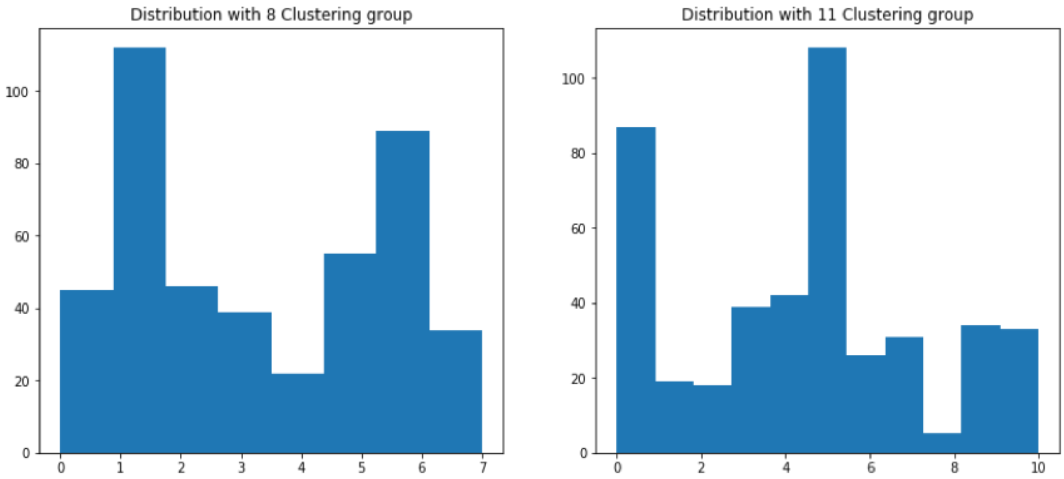

FIGURE 4.4: Prediction Distribution Of KMeans

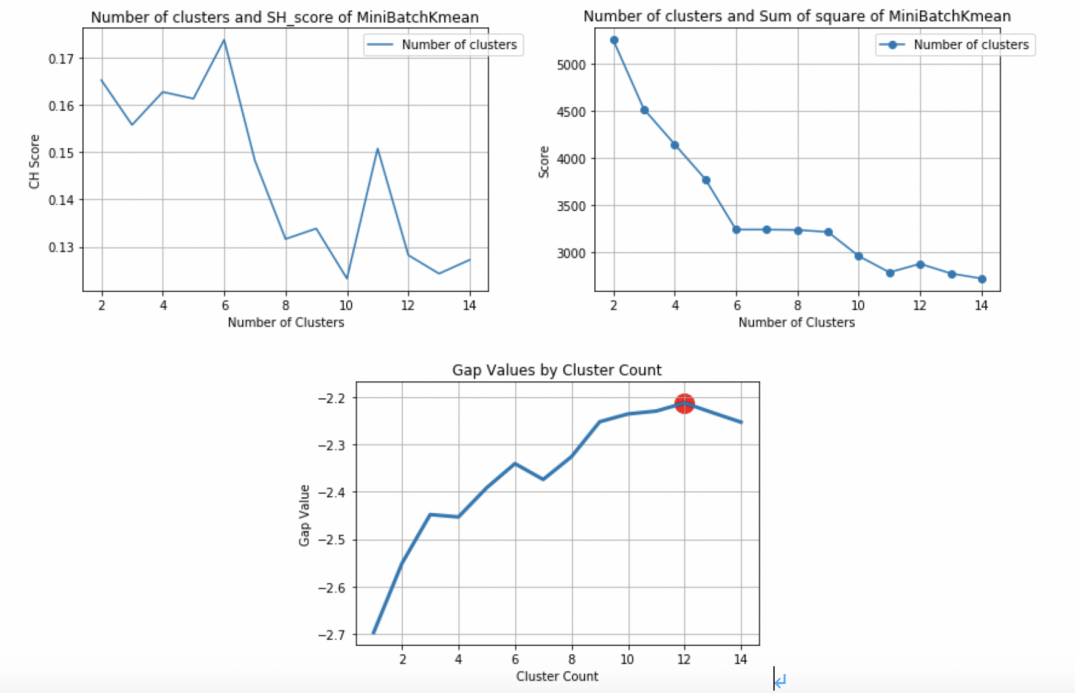

FIGURE 4.5: Three Evaluation Method for MiniBatchkmeans

objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. So for this model we just briefly present all the results below.

4.2.1 Modeling Evaluation Exploration

The model performance based on three evaluation method (Figure 4.5).

4.2.2 Prediction Distribution

For MinibatchKmeans we have a more clear result. We could see 6 and 11 as good cluster numbers and we use that three to see the prediction result (Figure 4.6). We could have a relative balance result.

4.2.3 Outliers Assignment

And the last, reassign outliers back to groups based on the distance metrics the same as KMeans. Detailed Analysis presented in the see next Chapter.

4.2.4 Model Conclusion Of KMeans And MiniBatchkmeans

KMeans model works well on the dataset after PCA and without outlier, and present a relative balanced prediction distribution with 8 and 11 clusters.

While MinibatchKmeans, based on the same algorithm as Kmeans, optimising the same objective function, shows more clear result of performance evaluation. And it presents a relative balanced prediction distribution with 6 and 11 clusters.

And it worth to mention that comparing and MiniBatchKMeans the former always have better performance under all the evaluation scores. We could see that

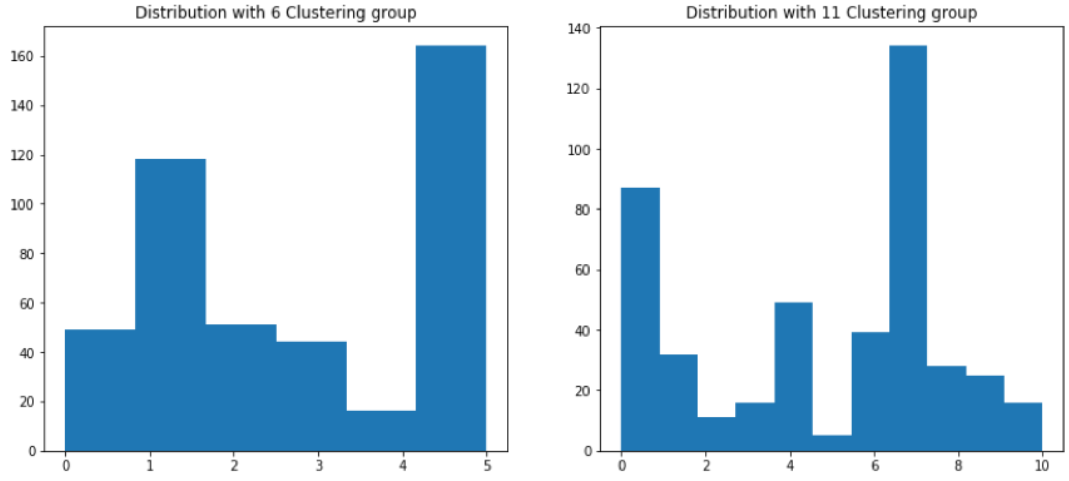

FIGURE 4.6: Prediction Distribution Of MiniBatchkmeans

MiniBatchKMeans converges faster than KMeans, but the quality of the results is reduced.

4.3 Birch

4.3.1 Searching For Optimal Hyper Parameters

The Birch algorithm has two parameters, the threshold and the branching factor. The branching factor limits the number of subclusters in a node and the threshold limits the distance between the entering sample and the existing subclusters.

Conduct Grid-search to find optimal parameters and then see the clustering result based on Calinski Carabaz score(CH score) performance evaluation. The Grid-search function code attached in appendix.

4.3.2 PCA Effect Evaluation

Based on CH score evaluation method we could also test if PCA improve data structure in Birch model. Figure 4.7 show optimal result of after PCA dataset and before PCA dataset.

In each case with same parameters, dataset after PCA has general higher CH scores than that before PCA. And Figure 4.7 only show the optimal parameter case of those two datasets. So we could conclude that PCA improve the data structure for Birch model.

4.3.3 Prediction Distribution

We continue the research dataset with after PCA. The relative optimal clustering number of after PCA is 4 and 8 with threshold = 1 and branchingfactor = 50. And the distribution graphs (Figure 4.8) show the prediction of 8 clusters has a more balanced distribution than that with 4 clusters.

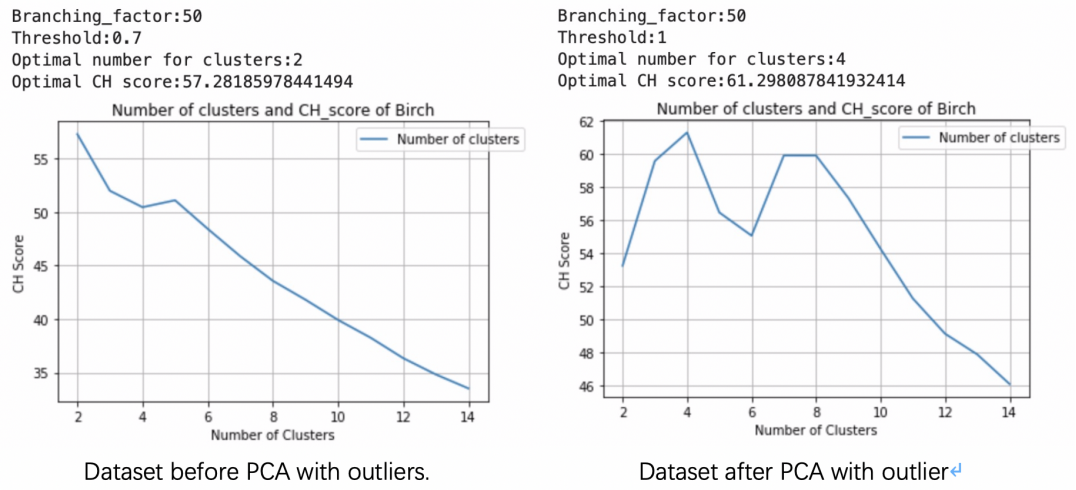

FIGURE 4.7: PCA Effect Evaluation Of Birch

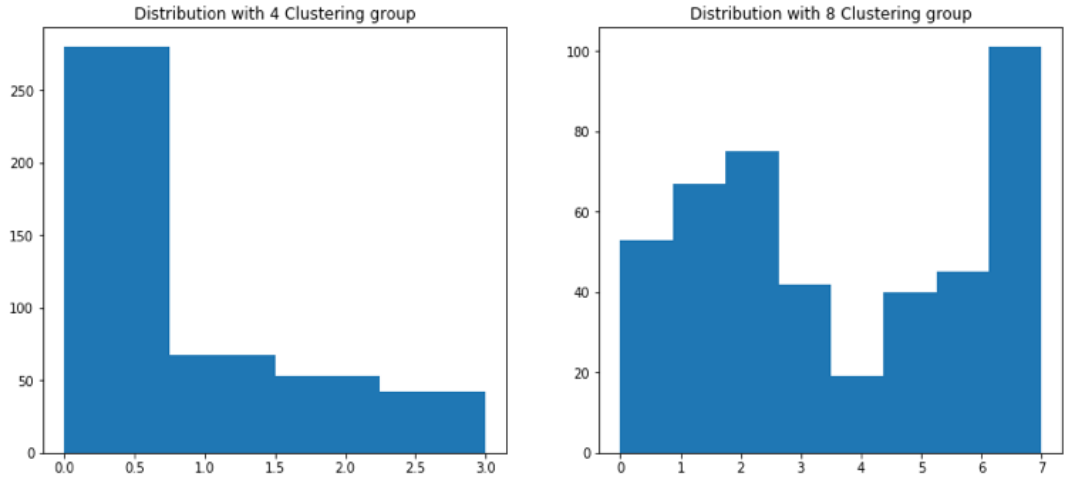

FIGURE 4.8: Prediction Distribution Of Birch

4.3.4 Outliers Assignment

We remove outlier and find the relative clustering numbers above. Next we reassign those outlier based on the distance between each outlier and each cluster center. After assignment, the CH score decrease, which is the same as KMeans and MinibatchKMeans.

4.3.5 Model Conclusion

Birch does not scale very well to high dimensional data. As a rule of thumb if features is greater than twenty, it is generally better to use MiniBatchKMeans. If the number of instances of data needs to be reduced, or if one wants a large number of subclusters either as a preprocessing step or otherwise, Birch is more useful than MiniBatchKMeans.

4.4 Agglomerative

For agglomerative clustering, there are three parameters could be changed: number of clusters, linkage type ('ward','complete', 'average', 'single'), and distance calculation methods ('euclidean','l1','l2','manhattan','cosine','precomputed'). We performed grid search for each type of dataset based on these three parameters with number of clusters changing from 2 to 20. For each cluster number, we find out the parameter combination that generates the highest Calinski-Harabasz (CH) score, since it is regarded as the primary evaluation method for agglomerative clustering model. A plot of CH score and Silhouette (SH) score versus number of clusters will be generate for each dataset. Moreover, SH scores are rescaled by multiply 100 for more intuitive comparison purpose.

4.4.1 Dataset before PCA

The result suggests that for dataset before PCA, the optimal combination for all cluster numbers happen to be linkage = 'ward' and affinity = 'euclidean'. As figure 4.9 suggested, the CH score for this dataset is descending as number of clusters increase. According to [14], descending or ascending or smooth-horizontal CH score doesn't provide any reference value – we cannot prefer one solution to others based on a line like this. Hence, we choose the optimal number of clusters through finding the peak for SH score.

As Figure 4.9 suggested, there are two abrupt elbows in SH score: cluster number equals to 3 and 12. Since we want an outcome with around 10 clusters, the optimal number of clusters for this dataset is therefore 12. According to Figure 4.10 and 4.11, it can be seen that the clusters are imbalanced.

4.4.2 Dataset after PCA, with outliers

With dimensionality reduction, the optimal configuration is still linkage = 'ward' and affinity = 'euclidean'. As Figure 4.12 suggested, the optimal number of clusters will be 9 for dataset after PCA. The imbalanced clusters problem still occurs (Figure 4.13, 4.14). One thing can be noticed is that the outcome after PCA is overall better than results before PCA: CH score increases around 40 % for all number of clusters.

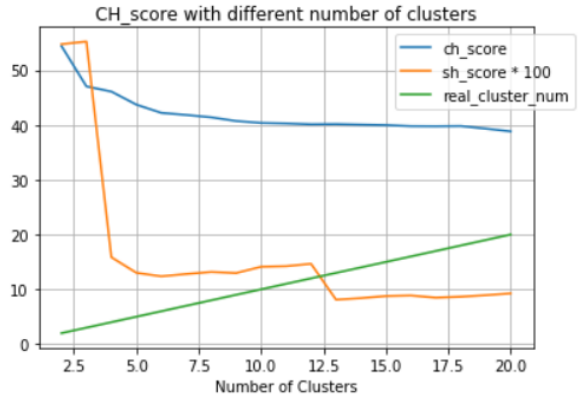

FIGURE 4.9: Agglomerative Clustering scores(before PCA)

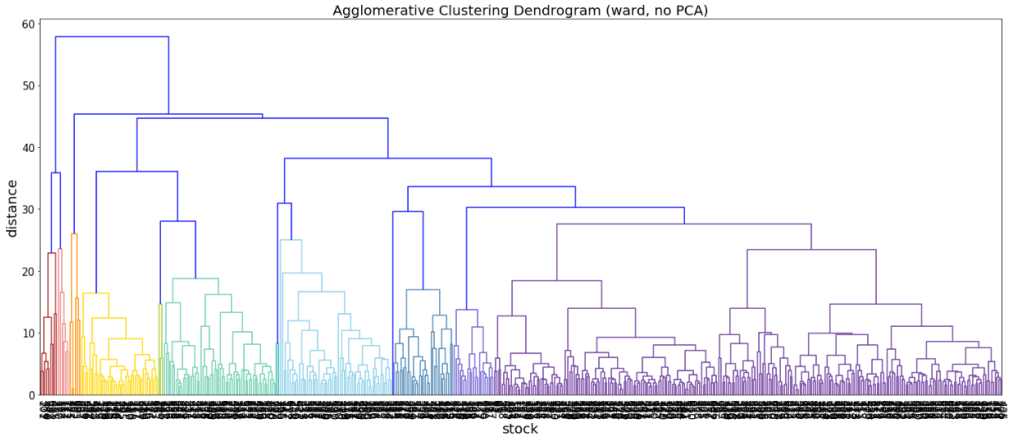

FIGURE 4.10: Dendrogram for Agglomerative Clustering Result (before PCA)

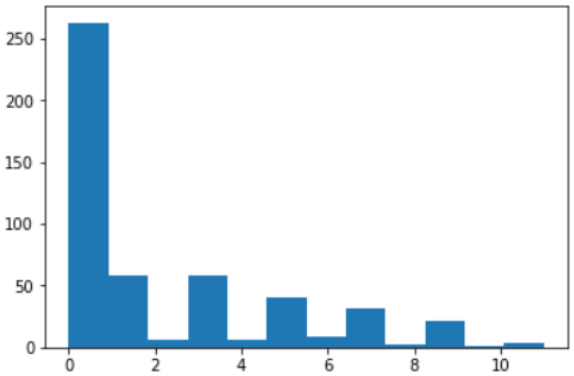

FIGURE 4.11: Histogram for Agglomerative Clustering Result (before PCA)

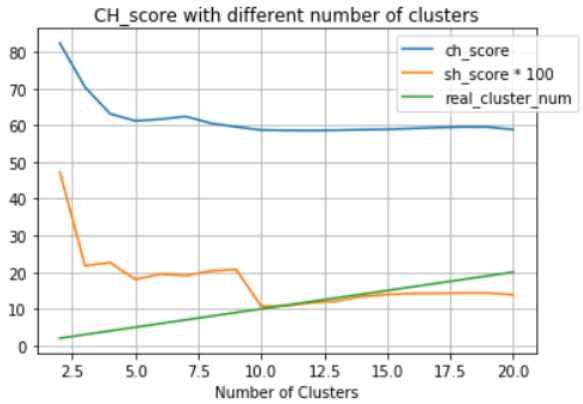

FIGURE 4.12: Agglomerative Clustering Scores (after PCA, with outliers)

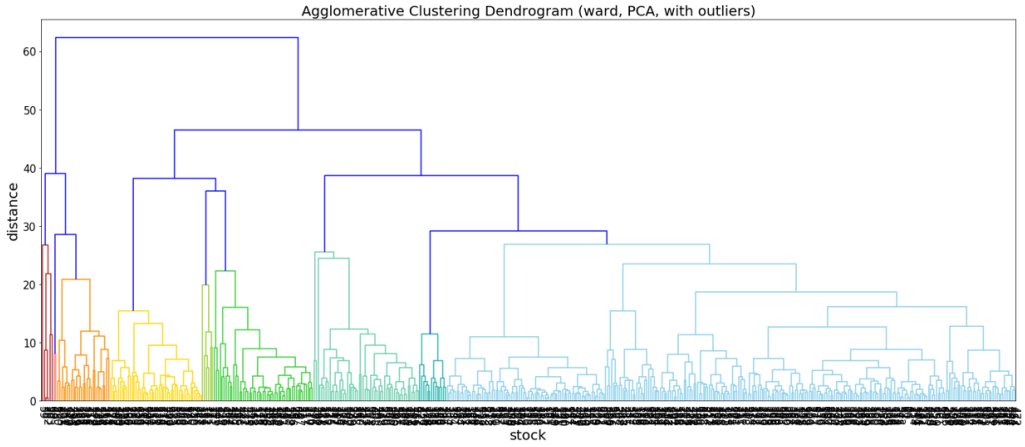

FIGURE 4.13: Dendrogram for Agglomerative Clustering Result (after PCA, with outliers)

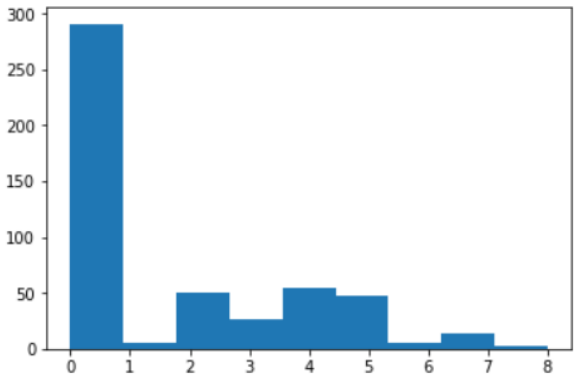

FIGURE 4.14: Histogram for Agglomerative Clustering Result (after PCA, with outliers)

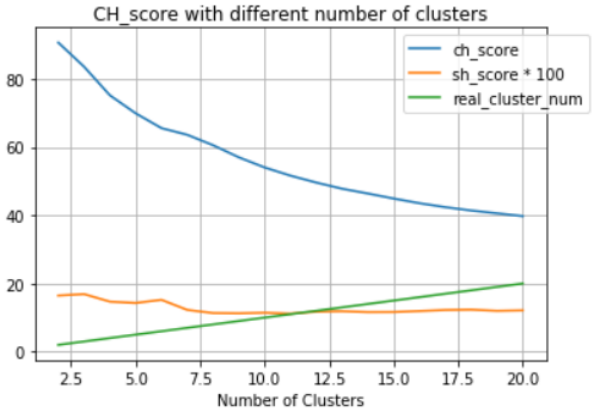

FIGURE 4.15: Agglomerative Clustering scores (after PCA, without outliers)

TABLE 4.1: Agglomerative Clustering Result Comparison

	# Clusters	CH score	SH score	Cluster_std
Before PCA (with outliers)	12	40.1227	0.1466	72.8629
After PCA (with outliers)	9	59.5859	0.2073	90.7277
After PCA (without outliers)	6	65.5629	0.1521	49.0252
After PCA (without outliers)	13	47.7946	0.1187	23.7171

4.4.3 Dataset after PCA, without outliers

After removing outliers, the combination of linkage = 'ward' and affinity = 'euclidean' still outperform the other combinations. While no abrupt elbow is noticed, there is a small peak for SH score when cluster number = 6 and 13 (Figure 4.15).

Both results generate relatively balanced clusters (Figure 4.16, 4.17).

4.4.4 Outlier Assignment

Then, outliers are labeled and added back to the whole dataset based on the clustering results of above two models. The clusters generated are still relatively balanced compared to the result before PCA and after PCA with outliers (Figure 4.18).

4.4.5 Model Conclusion

A more intuitive comparison can be seen from Table 4.1. We can conclude, for agglomerative clustering, using dataset after PCA without outliers will keep CH and SH score in acceptable range while generate relatively balanced clustering results. While it may be computationally expensive, Agglomerative clustering performs well with large number of samples and clusters. Moreover, this algorithm is also commonly used for unsupervised learning for its unique computation mechanism. In each successive iteration, it agglomerates the closest pair of samples by satisfying certain similarity criteria [15]. Because of this, outliers have a significant impact on the results. That is also the reason why after removing the outliers, the model generates more balanced clusters.

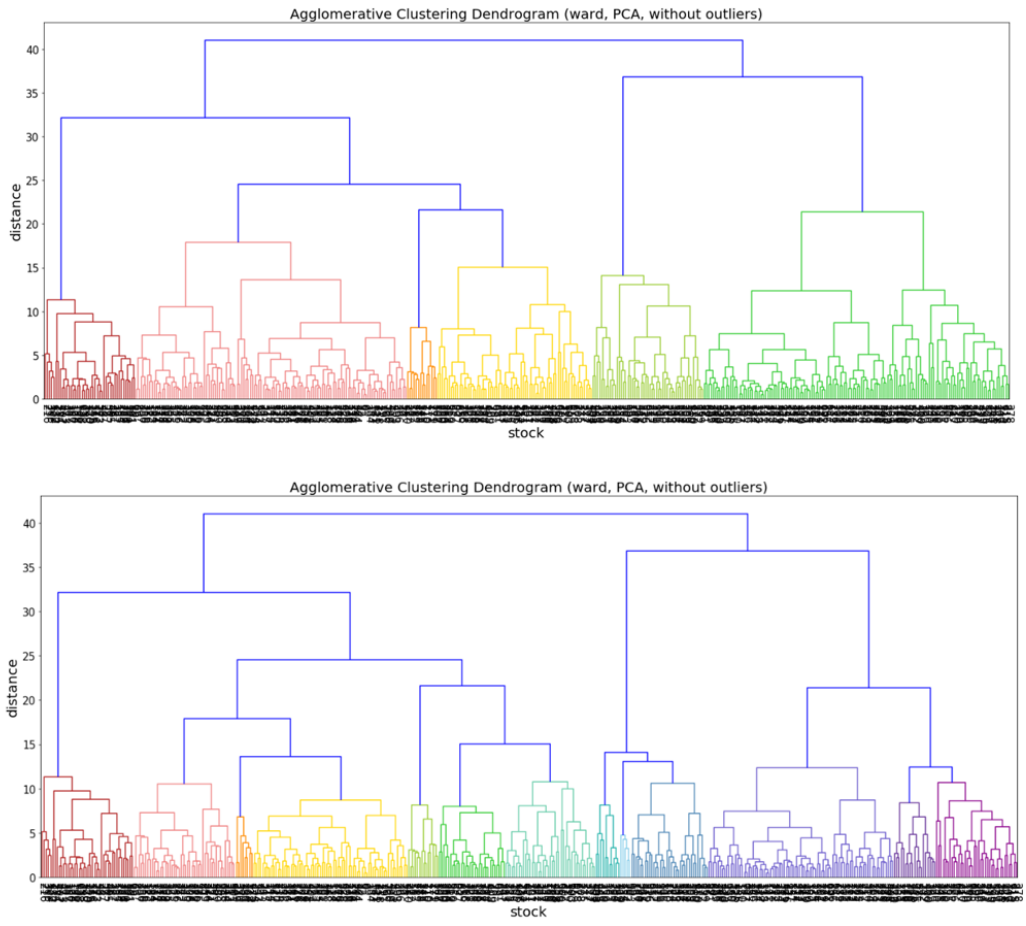

FIGURE 4.16: Dendrogram for Agglomerative Clustering n=6, 13 (after PCA, without outliers)

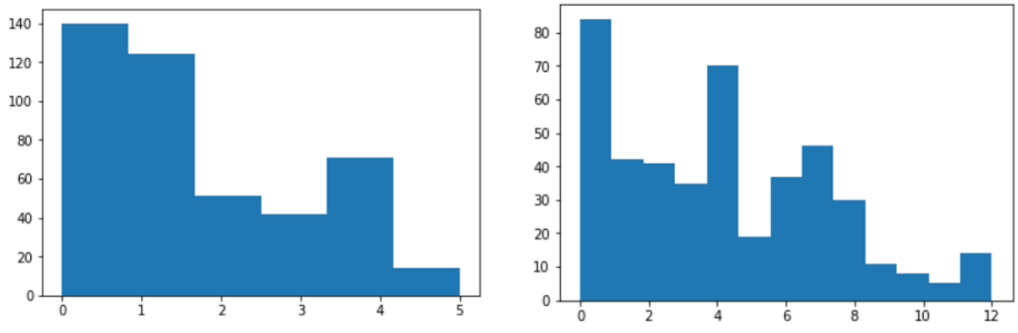

FIGURE 4.17: Histogram for Agglomerative Clustering n=6, 13 (after PCA, without outliers)

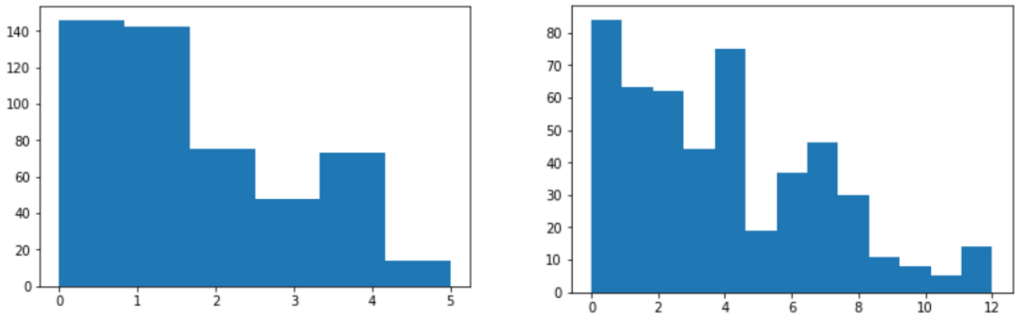

FIGURE 4.18: Histogram for Agglomerative Clustering n=6, 13 (Placing back outliers)

TABLE 4.2: Spectral Model Parameters and Explanation

Name	Description
n_clusters	Number of clusters
n_init	Number of time the k-means algorithm will be run with different centroid seeds. $np.arange(5, 30, 3)$
n_neighbors	Number of neighbors to use when constructing the affinity matrix using the nearest neighbors method. $np.arange(5, 30, 5)$
Gamma	Kernel coefficient for rbf, poly, sigmoid, laplacian and chi2 kernels. $np.arange(0.6, 1.5, 0.2)$
affinity	'nearest_neighbors', 'precomputed', 'rbf'

4.5 Spectral

For Spectral Clustering, there are five parameters that can be changed (Table 4.2). We tested the algorithm by ranging number of clusters from 2 to 20. As the customized grid search function for Agglomerative Clustering, we traversing all combinations to find out the one that generates the highest CH score for certain number of clusters.

4.5.1 Dataset before PCA

For dataset before PCA, there is a peak in both CH and SH score when number of clusters equals to 4 (Figure 4.19), n_init = 5, gamma = 0.6, affinity = nearest_neighbors, n_neignbors = 10. This might because spectral clustering performs the best when given medium number of samples and small number of clusters [5]. However, the clustering result (Figure 4.20) suggests that the clusters generated are imbalanced.

To satisfy our clustering analysis purpose, more clusters is desired as we mentioned in the beginning of this thesis. We found another peak in SH score when number of clusters = 9, n_init = 10 , gamma = 0.6 ,affinity = nearest_neighbors , and n_ neighbors = 10. The clusters are slightly more balanced than number of clusters equals to 4(Figure 4.20).

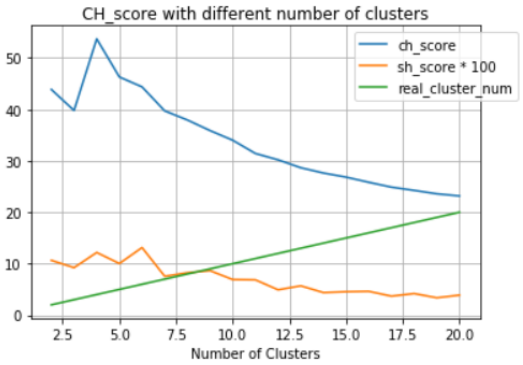

FIGURE 4.19: Spectral Clustering Model scores (before PCA)

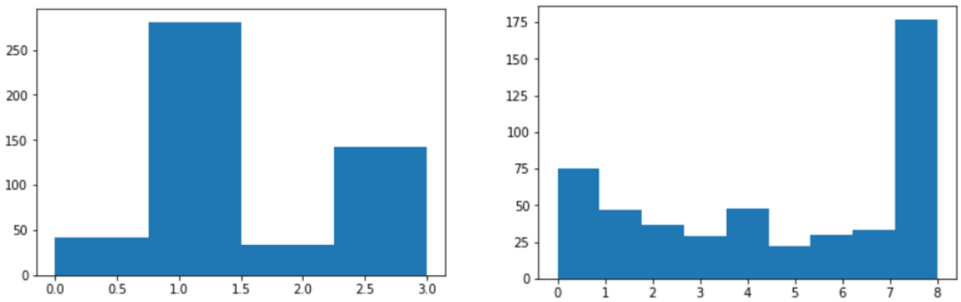

FIGURE 4.20: Histogram for Spectral Clustering n=4, 9 (before PCA)

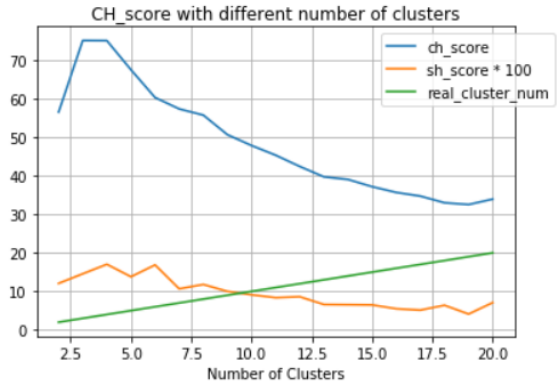

FIGURE 4.21: Spectral Clustering Model scores (after PCA, with outliers)

4.5.2 Dataset after PCA, with outliers

After PCA and before dealing with outliers, optimal combinations vary for different number of clusters. The overall CH scores increased compared with the outcome before PCA (Figure 4.21).

As above situation, a peak for both CH and SH score can be noticed when number of clusters = 4, n_init = 5, gamma = 0.6, affinity = nearest_neighbors, and number of neighbors = 10. In this case (Figure 4.22), the clusters are relatively imbalanced. Another two sets of configurations are also regarded as acceptable as they capture the peak in SH score. That is when number of clusters = 8, n_init = 20, gamma = 0.6, affinity = nearest_neighbors, number of neighbors = 15 and when number of clusters = 12, n_init = 10, gamma = 0.6, affinity = nearest_neighbors, number of neighbors = 15. As the number of clusters increased, the clustering outcome is getting more balanced.

4.5.3 Dataset after PCA, without outliers

After removing the outliers from dataset with PCA, the CH score increased significantly for small number of clusters (Figure 4.23). One small peak in both CH and SH score appears when number of clusters = 8, n_init = 5, gamma = 0.6, affinity = nearest_neighbors, and number of neighbors = 25.

The result is beautifully balanced (Figure 4.24).

4.5.4 Outlier Assignment

After labelling and adding back the outliers, the result is still balanced (Figure 4.25). Most of the outliers fall in the first three clusters. The overall shape of the clustering distribution doesn't change obviously.

4.5.5 Model Conclusion

A more intuitive comparison can be seen from Table 4.3. The dataset after PCA without outliers performs the best as well. We can also conclude that generally, Spectral Clustering algorithm works better for small number of clusters.

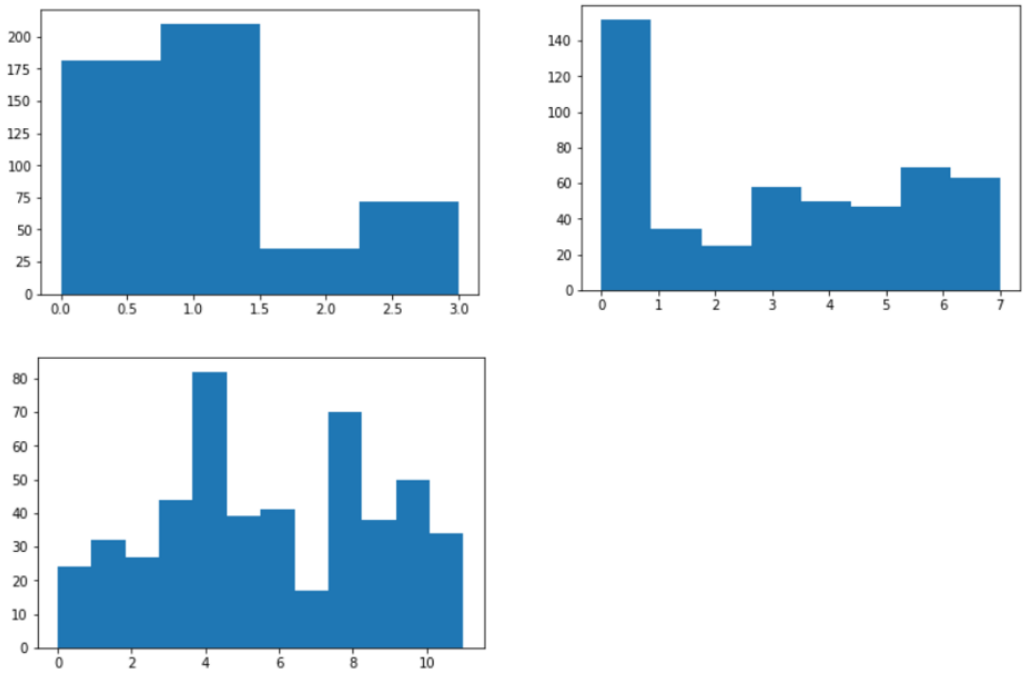

FIGURE 4.22: Histogram for Spectral Clustering n=4, 8, 12 (after PCA, with outliers)

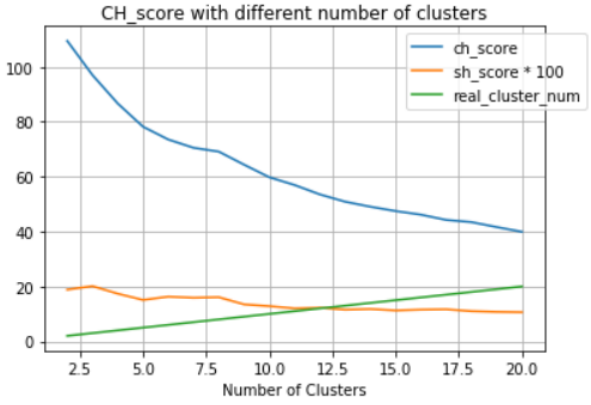

FIGURE 4.23: Spectral Clustering Model scores (after PCA, without outliers)

TABLE 4.3: Spectral Clustering Result Comparison

	# Clusters	CH score	SH score	Cluster_std
Before PCA	4	53.7176	0.1218	115.4369
Before PCA	9	35.8770	0.0864	48.2105
After PCA (with outliers)	4	75.0881	0.1703	84.2002
After PCA (with outliers)	8	55.7545	0.1179	39.0741
After PCA (with outliers)	12	42.4231	0.0860	18.6182
After PCA (without outliers)	8	69.1317	0.1609	24.9967

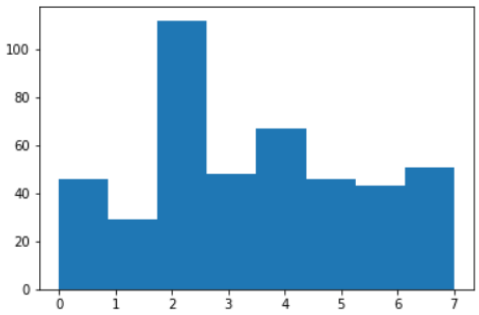

FIGURE 4.24: Histogram for Spectral Clustering (after PCA, without outliers)

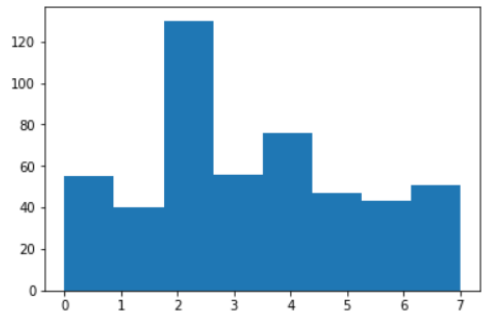

FIGURE 4.25: Histogram for Spectral Clustering (Placing back outliers)

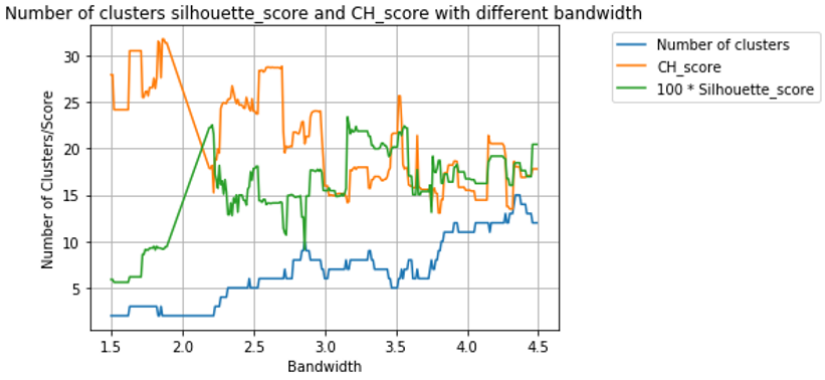

FIGURE 4.26: Mean Shift Algorithm scores(before PCA, with outliers)

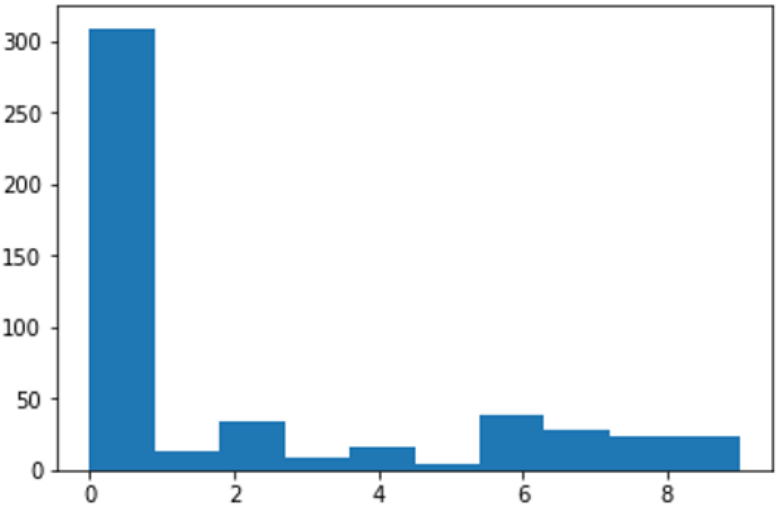

FIGURE 4.27: Histogram for Mean Shift Clustering(before PCA, with outliers)

4.6 Mean Shift

As mentioned in the background part, Mean Shift Algorithm has so many merits that make it a robust model. We expect the results to be good and let's try our four datasets. What's more, there is only one hyper parameter needed to be tuned (bandwidth).

4.6.1 Dataset before PCA with outliers

As presented in the figure 4.26, the results of Mean Shift model look fine when the bandwidth goes above 2.5: The number of clusters are around 10 and the CH scores are acceptable as well.

However, when we focus on the number of the samples in different clusters, it is imbalanced as shown in Figure 4.27.

This result can serve as benchmark and we can compare the result of this dataset with those of the other three to test whether PCA and Outliers detection enhance model performance.

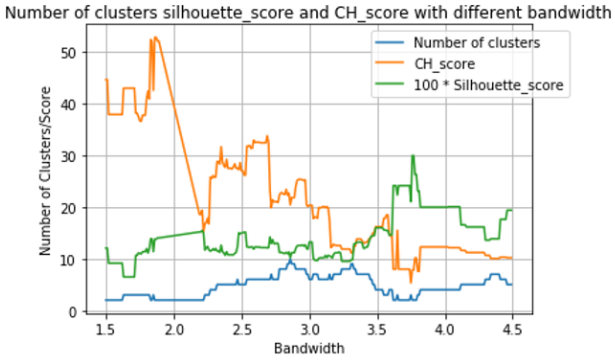

FIGURE 4.28: Mean Shift Algorithm scores(before PCA, without outliers)

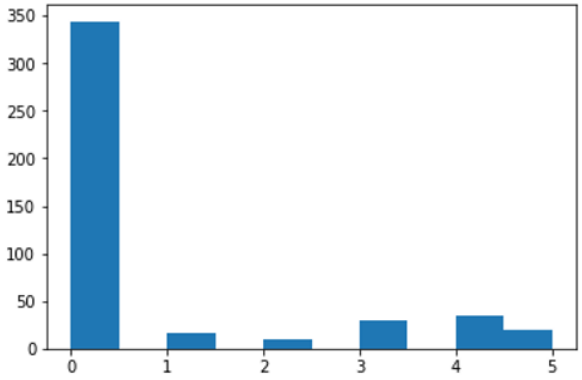

FIGURE 4.29: Histogram for Mean Shift Clustering Result(before PCA, without outliers)

4.6.2 Dataset before PCA without outliers

The result of this dataset is similar to last dataset. Dealing with outliers doesn't make the model performance better.(Figure 4.28)

No matter what value we pick for bandwidth, we always end up with imbalanced clusters.(Figure 4.29)

We can come to conclusion that removing outliers doesn't help us improve Mean Shift model with features before PCA.

4.6.3 Dataset after PCA with outliers and without outliers

Similar situation happens in these two datasets. Although CH score rises compared to model without PCA. The results are still imbalanced.(Figure 4.30, 4.31)

4.6.4 Model Conclusion

In all, Mean Shift model doesn't work well on the features no matter if we use PCA or Outliers Removal method.

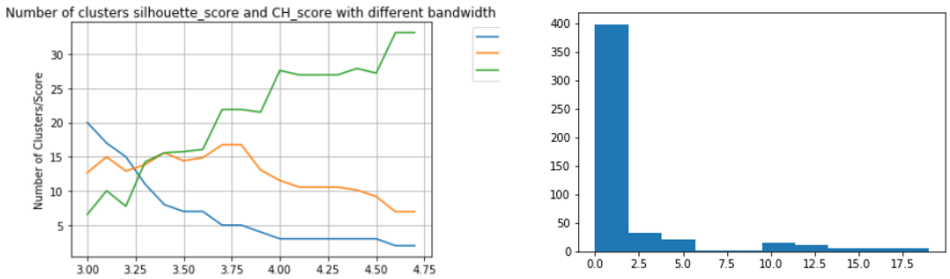

FIGURE 4.30: Mean Shift Clustering Result (After PCA, with outliers)

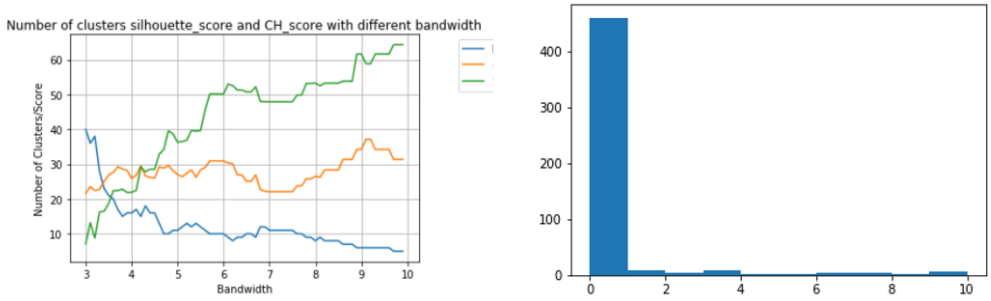

FIGURE 4.31: Mean Shift Clustering Result (After PCA, without outliers)

4.7 DBSCAN

DBSCAN is derived from spatial clustering and it is density-based and able to deal with noise. However, the results are similar to spatial clustering. DBSCAN performs badly for all distance matrixes. The major problem is imbalanced clusters as well.

For distance matrix, Cityblock, Cosine, Euclidean, L1, L2 and Manhattan are tested. And there are two hyper parameters need to be tuned. Grid Search is used and we filter those results have around 10 clusters and relatively high CH score.

However, the distributions of the number of the samples in clusters are imbalanced shown in the figure 4.32.

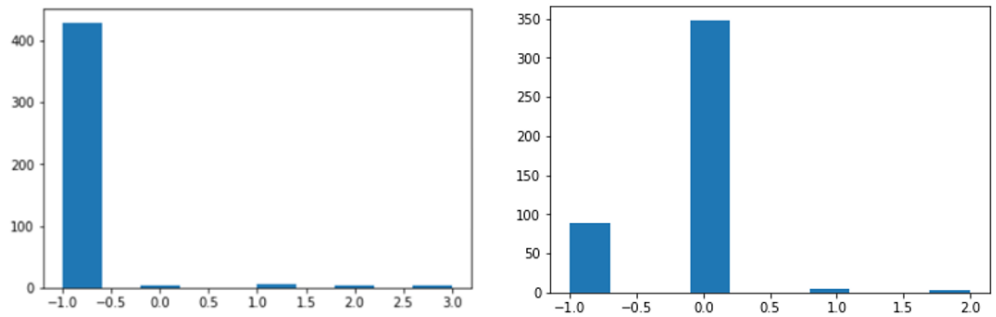

FIGURE 4.32: DBSCAN Clustering Result

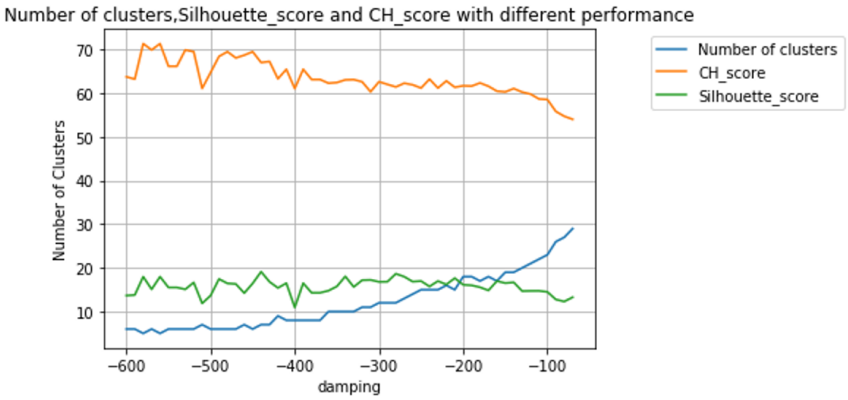

FIGURE 4.33: Affinity Propagation Clustering Scores (Before PCA, with outliers)

DBSCAN requires the density of the samples invariant within each cluster. This assumption might not hold for either of our four datasets. That could be reason DBSCAN does not work.

4.8 Affinity Propagation

Preference is a crucial hyper parameter in Affinity Propagation. The number of clusters varies as preference changes.

4.8.1 Dataset before PCA with outliers

We still start with dataset before PCA with outliers. The results are much better than those of DBSCAN and Mean Shift. CH score is around 60 and the outcomes are balanced. (Figure 4.33)

However, the only drawback is that some of clusters only have one or two samples. The outliers have negative effects on the Affinity Propagation algorithm.(Figure 4.34)

This dataset without PCA and outliers can serve as benchmark to the other three datasets. We can see how PCA and Outliers Detection effect the results.

4.8.2 Dataset before PCA without outliers

Classify samples without outliers

We remove outliers according to 5-sigma rule and train the model. Similar to the benchmark, we also get balanced results with high CH score. (Figure 4.35)

One improvement is that there is not any cluster only with one or two samples. All clusters contain at least 10 samples, which means our model is not creating clusters just for the outliers.(Figure 4.36)

We picked -150 as the final value for hyper parameter, preference, considering the number of clusters and CH score. The CH score is 43 and the number of clusters is 9 for this case.

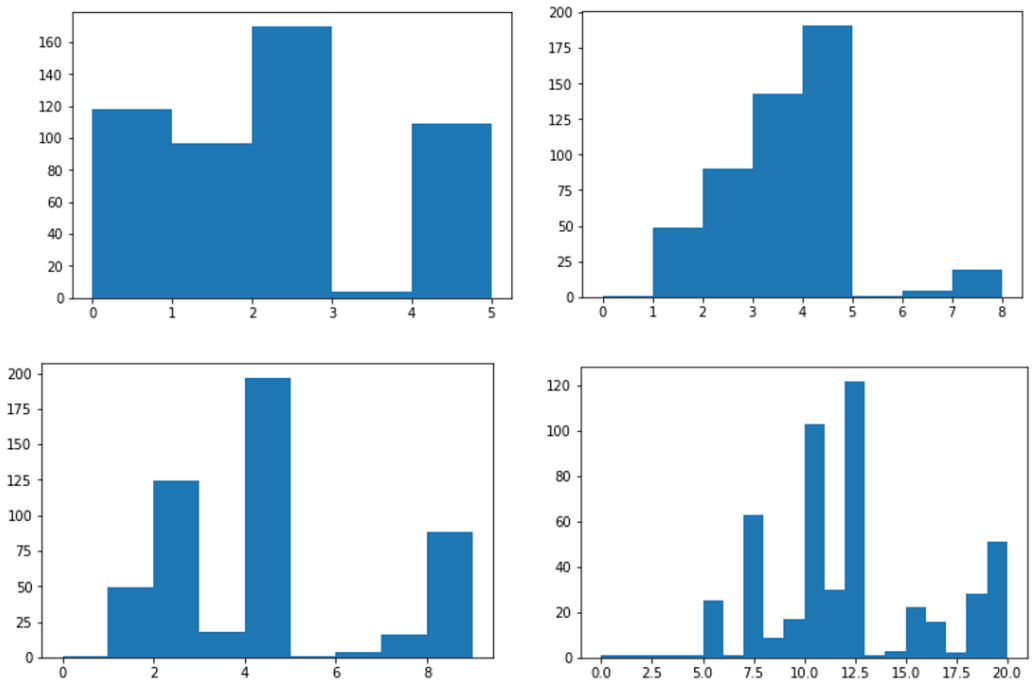

FIGURE 4.34: Histogram for Affinity Propagation Clustering Result (Before PCA with outliers)

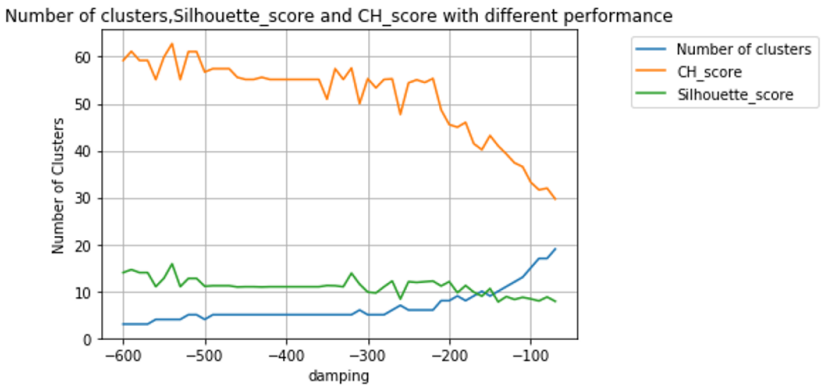

FIGURE 4.35: Affinity Propagation Clustering Scores (Before PCA, without outliers)

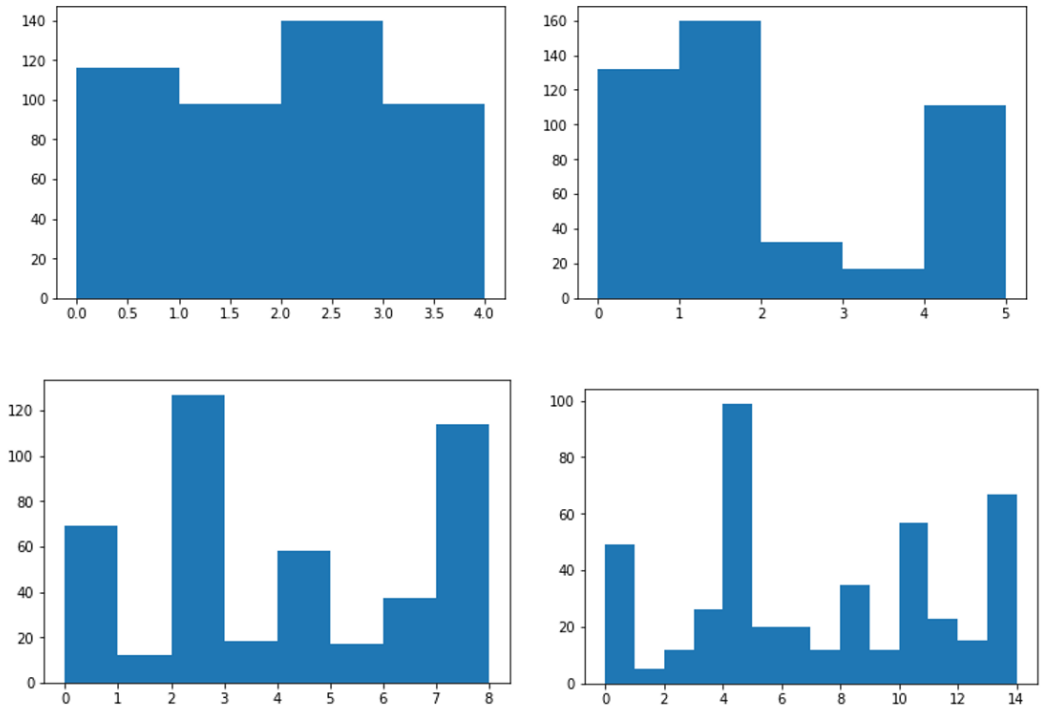

FIGURE 4.36: Histogram for Affinity Propagation Clustering Result (Before PCA without outliers)

Assign outliers back to clusters

Then we assign the outliers back to clusters according to the distance between them and centers of the clusters. For each outlier, we calculate the distances between outlier and each clustering center. Then find the shortest distance and assign the outlier to that clustering center.

After assigning the outliers back to clusters. CH score decreases significantly, from 43 to 31, which indicates that the outliers do not fit in any clusters.

4.8.3 Dataset after PCA with outliers

After PCA transformation, CH score remain unchanged compared with benchmark.(Figure 4.37)

And for the distribution of the samples, there are 5 clusters that only contain less than 3 samples. These clusters are true clusters, just holders for outliers. (Figure 4.38)

4.8.4 Dataset after PCA with outliers

Classify samples without outliers

After PCA and Outliers removal, the CH score is much better than the benchmark. PCA and Outliers removal bring significant positive effects to the model.(Figure 4.39)

And as shown in the figure 4.40, the number of samples in each cluster is balanced and there are no fake cluster contain only one or two sample. These are quite satisfying result.

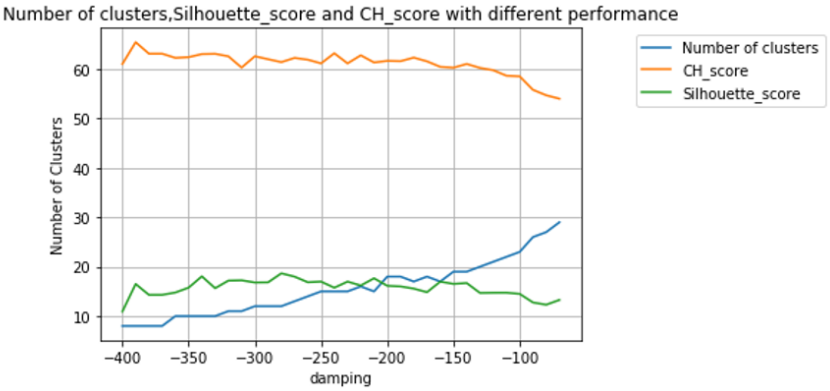

FIGURE 4.37: Affinity Propagation Clustering Scores (After PCA, with outliers)

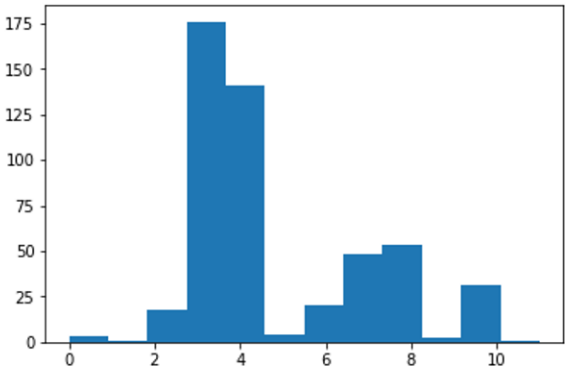

FIGURE 4.38: Histogram for Affinity Propagation Clustering Result (After PCA without outliers)

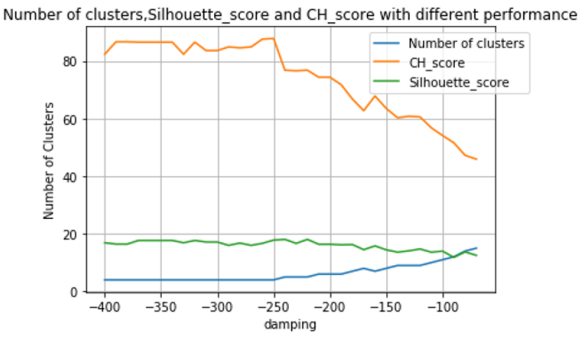

FIGURE 4.39: Affinity Propagation Clustering Scores (After PCA, without outliers)

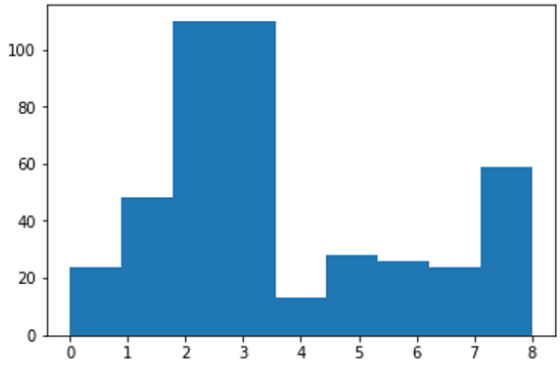

FIGURE 4.40: Histogram for Affinity Propagation Clustering Result (After PCA without outliers)

Assign outliers back to clusters

We assign outliers back to clusters, but CH score decrease by 20, from 80 to 60.

4.8.5 Model Conclusion

To sum up, Affinity Propagation work well in these datasets and beats DBSCAN and Mean Shift. Outliers removal can improve the model performance and can eliminate fake clusters that only have one or two samples. Among the four datasets, the one with both PCA and outliers removal work the best: balanced result with 80 CH score.

Chapter 5

Model Comparison

5.1 The distribution of the results

According to the evenness of the results of the models, the clustering methods can be classified into two groups:
1) The models with balanced results: Affinity Propagation, K-Mean, MiniBactch K-Mean, Birch, Spectral and Agglomerative.
2) The models with imbalanced results: Mean Shift, DBSCAN.

For those models that have reasonable results, we try to rank them by using Silouette Coefficient and CH score separately.

5.2 Silouette Coefficient and CH score Comparisons

In order to have a better idea of the results of the models, we rank the best model of each clustering method according to silhouette scores. Best models should be models that have balanced results and around ten clusters. We show the ranking in the Table 5.1.

The differences among models Silhouette Scores are not that significant. However, we can't jump into conclusion that Agglomerative is the best model and Birch is the worst among these models.

Then we turn to see CH scores of different models.(Table 5.2)

Combining CH score and Silhouette score, we can say that Birch works the worst in these models. Agglomerative clustering and Spectral clustering work better than others.

TABLE 5.1: Model Silhouette Score Comparison

Clustering Methods	**Silhouette Score**
Agglomerative	0.21
Kmean	0.18
Spectral	0.16
MiniBatch-Kmean	0.15
Affinity Propagation	0.15
Birch	0.13

TABLE 5.2: Model CH Score Comparison

Clustering Methods	CH Score
Spectral	69
Affinity Propagation	61
Agglomerative	60
Kmean	58
MiniBatch-Kmean	53
Birch	51

5.3 PCA transformation effects and outliers removal effects

For features after PCA, each model gets higher CH score and Silhouette score, usually 10% to 30% higher. This convinces us that PCA transformation can help clustering models to correctly classify the samples.

For outliers removal, it is also proven a necessary process in data processing. If dataset with outliers is used, some clusters with only a few samples (under 5 samples) will be created for outliers. These clusters only contain some samples that are so special that can't be assigned to other clusters. However, after outliers detection and removal, there are no fake clusters created anymore. Every cluster has over 20 samples.

In all, PCA transformation and outliers removal are beneficial methods that can improve models performance and clustering results.

5.4 Outliers Reassignment

After we finish clustering the dataset without outliers, we assign the outliers we removed back to the clusters and we have a look at CH score and silhouette score. As expected, they drop accordingly. However, the drop is acceptable. Outliers certainly will have negative effects on the scores because they do not belong to any of the existing clusters. The reason we still want to assign them back is that If we have to give a label to these abnormal samples, we still can give them labels and the process of giving them labels will not effect we classify those normal samples.

Chapter 6

Conclusion and Future Research

In our capstone project, data is retrieved from Yahoo Finance. We clean the data, deal with outliers and do the feature engineering. After that, PCA analysis and transformation are used to perform dimensions reduction. Then, data is fed into different clustering models. After we tune the hyper parameters in the models, we find out which models can classify the samples correctly without imbalanced issue. Also, Evaluation with CH score and silhouette score are also hired to rank the models. There are still works left needed to be done. What's more, we manage to figure out how PCA and outliers removal improve our models' performance.

However, there are still works left to be done. There are still room for us to improve the features. More work about the features engineering can be done. Since we only have 6 features for technical features, more technical features can be tested and added to framework. Besides, the number of fundamental features is much greater than that of technical features. This makes fundamental features more important than technical features when we do the training. Weights can be assigned to make them equally important.

What's more, Equivalency test and Distance test can be performed in future research. Equivalency test can assist to find the most similar products within a universe. Distance analysis can be utilized to determine the effectiveness of the equivalency test. In addition, we can combine the results of previous clustering methods and try to test the consistency of the results. Question like are there some pairs of stocks always been clustered into one group can be solved.

Appendix A

Code: Data Preprocessing

A.1 Code: Get Data

```
# regular imports
import pandas as pd
import numpy as np

# for stock features import
import yahoo_historical
import yahoofinancials
from yahoofinancials import YahooFinancials

# get ticker table
ticker_url = "https://en.wikipedia.org/wiki/List_of_S%26P_500_companies"
ticker_table = pd.read_html(ticker_url)

# generate tickers list
tickers = ticker_table[0][0][1:].tolist()

# get technical data
# get the historical price data
sp_data = YahooFinancials(tickers)
sp_prices = sp_data.get_historical_price_data('2018-06-01', '2019-06-01', 'daily')

result = []
for stock in tickers:
    # check whether we got the prices data for a pecific stock
    # if not, pass to the next
    if "prices" not in sp_prices[stock]:
        continue

    temp_list = sp_prices[stock]["prices"]

    # reformatting the list
    for daily_data in temp_list:
        result_temp = []
        result_temp.append(stock)
        result_temp.append(daily_data["formatted_date"])
        result_temp.append(daily_data["high"])
        result_temp.append(daily_data["low"])
        result_temp.append(daily_data["adjclose"])
        result_temp.append(daily_data["volume"])
        result.append(result_temp)

# store the data into pandas dataframe
sp_df = pd.DataFrame(np.array(result))
sp_df.columns = ["ticker","date","high","low","adjclose","volume"]
```

```
45
46 # output to csv file
47 sp_df.to_csv(r'sp500.csv')
48
49 # get fundamental data
50 # get the key statistics of the stocks
51 sp_fundamental = sp_data.get_key_statistics_data
52
53 df_fundamental = pd.DataFrame.from_dict(sp_fundamental, orient = 'index
       ')
54
55 sp_eps = sp_data.get_earnings_per_share()
56 sp_mc = sp_data.get_market_cap()
57 sp_pe = sp_data.get_pe_ratio()
58 sp_ni = sp_data.get_net_income()
59 sp_ebit = sp_data.get_ebit()
60
61 df_eps = pd.DataFrame(sp_eps, index=['eps']).T
62 df_mc = pd.DataFrame(sp_mc, index=['Market Cap']).T
63 df_pe = pd.DataFrame(sp_pe, index=['PE']).T
64 df_ni = pd.DataFrame(sp_ni, index=['NI']).T
65 df_ebit = pd.DataFrame(sp_ebit, index=['EBIT']).T
66
67 df = df_fundamental.join(df_eps, how = 'outer')
68 df = df.join(df_mc, how = 'outer')
69 df = df.join(df_pe, how = 'outer')
70 df = df.join(df_ni, how = 'outer')
71 df = df.join(df_ebit, how = 'outer')
72
73 # output to csv file
74 df.to_csv(r'sp500_fundamental_eps_mc.csv')
```

A.2 Code: Feature Engineering

```
import pandas as pd
import numpy as np

from sklearn.pipeline import Pipeline
try:
    from sklearn.impute import SimpleImputer # Scikit-Learn 0.20+
except ImportError:
    from sklearn.preprocessing import Imputer as SimpleImputer

from sklearn.preprocessing import StandardScaler
from sklearn.base import BaseEstimator, TransformerMixin
from scipy.stats import skew, mstats
from scipy.stats import kurtosis

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

data_f = pd.read_csv(r"Fundamental_features.csv",index_col=0)
# f for Fundamental
data_t = pd.read_csv(r"Tech_Features.csv",index_col = 0)
# t for Tech

# Get SP500 index from yahoo finance for feature engineering purpose.

from yahoofinancials import YahooFinancials
yahoo_financials = YahooFinancials("^GSPC")
historical_stock_prices = yahoo_financials.get_historical_price_data('
    2018-06-01', '2019-06-01', 'daily')
sp500 = pd.DataFrame(historical_stock_prices["^GSPC"]["prices"])


# Impute Missing Values and Feature Engineering

# technical features
# find ticker short of dates
ticker_delete = []
tickers = data_t["ticker"].unique()
rst = data_t.groupby('ticker')['date'].count()
for ticker in tickers:
    if rst[ticker] != int(rst.median()):
        ticker_delete.append(ticker)

print("Stock with unmatching data:")
print(ticker_delete)
for ticker in ticker_delete:
    data_t =data_t.drop(data_t[data_t["ticker"]== ticker ].index, axis
    = 0)

# get close price, volume and return dataframes
close_df = data_t.pivot(index='ticker', columns='date',values='
    close_price')
volume_df = data_t.pivot(index='ticker', columns='date',values='volume'
    )
return_df = (close_df.pct_change(axis='columns')).drop(close_df.columns
    [0], axis=1)

# feature engineering for technical features
return_sp = ((sp500.adjclose.diff()/sp500.adjclose)[1:]).mean()
```

```
return_std = return_df.std(axis=1)
return_skew = return_df.skew(axis=1)
return_kurtosis = return_df.kurtosis(axis=1)
Max_Drawdown = (1 - close_df/close_df.cummax(axis = 1)).max(axis=1)
sharpe_ratio = (return_df.mean(axis=1)-return_sp)/return_std
volume_std = volume_df.std(axis=1)
t_feature = pd.DataFrame()
t_feature = t_feature.assign(Return_std = return_std, Return_skew =
    return_skew, Return_kurtosis = return_kurtosis, Sharpe_Ratio =
    sharpe_ratio, Volume_std = volume_std, Maxdrawdown = Max_Drawdown)


# fundamental features
# drop empty columns
data_f = data_f.dropna(axis='columns', how='all')
data_f = data_f[:-2]

# drop date columns and useless columns
data_f = data_f.drop(['fundInceptionDate','sharesShortPreviousMonthDate
    ', 'lastSplitDate', 'lastSplitFactor','priceHint', '
    dateShortInterest', 'maxAge', 'mostRecentQuarter',  '
    lastFiscalYearEnd', 'nextFiscalYearEnd'], axis=1)

# copy to data_f
f_feature = data_f.copy()

# reset the index
f_feature.index = data_f["Ticker"]
f_feature = f_feature.drop(["Ticker"],axis = 1)

# in case there are infinity entries
f_feature = f_feature.replace([np.inf, -np.inf,"infinity","Infinity"],
    np.nan)

# Feature Combination
# show the difference between two dataframe
t_inx = t_feature.index
f_inx = f_feature.index
display(t_inx.difference(f_inx))
display(f_inx.difference(t_inx))

feature_df = t_feature.join(f_feature, how='inner')

# Impute Missing Value and Standardization using Pipeline

class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values

features = feature_df.columns

Preprocess_pipeline = Pipeline([
        ("select_numeric", DataFrameSelector( features )), # set the
    parameter for DataDrameSelector
        ("imputer", SimpleImputer(strategy="mean")),
        ("StandardScaler", StandardScaler(copy=True, with_mean=True,
    with_std=True))

])
```

```
111
112 train_array = Preprocess_pipeline.fit_transform(feature_df)
113
114 train_df = pd.DataFrame(train_array)
115 train_df.index = feature_df.index
116 train_df.columns = feature_df.columns
117 train_df.head()
```

A.3 Code: Principal Components Analysis and Outlier Processing

```

# continue with Code: Feature Engineering

# Principal Components Analysis

def heatMap(df, mirror):

    # Create Correlation df
    corr = df.corr()
    # Plot figsize
    fig, ax = plt.subplots(figsize=(20, 20))
    # Generate Color Map
    colormap = sns.diverging_palette(220, 10, as_cmap=True)

    if mirror == True:
        # Generate Heat Map, allow annotations and place floats in map
        sns.heatmap(corr, cmap=colormap, annot=True, fmt=".2f")
        # Apply xticks
        plt.xticks(range(len(corr.columns)), corr.columns);
        # Apply yticks
        plt.yticks(range(len(corr.columns)), corr.columns)
        # show plot
    else:
        # Drop self-correlations
        dropSelf = np.zeros_like(corr)
        dropSelf[np.triu_indices_from(dropSelf)] = True
        # Generate Color Map
        colormap = sns.diverging_palette(220, 10, as_cmap=True)
        # Generate Heat Map, allow annotations and place floats in map
        sns.heatmap(corr, cmap=colormap, annot=True, fmt=".1f", mask=
    dropSelf)
        # Apply xticks
        plt.xticks(range(len(corr.columns)), corr.columns);
        # Apply yticks
        plt.yticks(range(len(corr.columns)), corr.columns)
    # show plot
    plt.show()

# draw heat map
heatMap(train_df,False)

pca_feature = PCA(random_state=100)
_ = pca_feature.fit(train_df)

acc_va_ratio = np.cumsum(pca_feature.explained_variance_ratio_)
# output variance ratio
acc_va_ratio

# draw variance ratio plot
PCA_var_ratio=plt.figure()
plt.plot(acc_va_ratio)
plt.legend(["Accumulated Variance Ratio"],bbox_to_anchor=(1.1,1))
plt.grid()
plt.title("Accumulated Variance Ratio")
plt.xlabel('Components')
plt.ylabel('Percentage')
plt.axhline(y=0.8)
_ = plt.show()
```

```
58
59 # set threshold to 0.8 and keep first 11 components
60 m_pca = PCA(n_components=11).fit_transform(train_df)
61
62
63 # Dealing with Outliers
64 # before PCA
65 # Without pca, samples with outliers:
66 outliers_tickers = np.array(train_df[(train_df>5)|(train_df<-5)].dropna
       (how = 'all',axis = 0).index)
67 data_o = train_df.loc[outliers_tickers,:]
68 # Without pca, samples without outliers
69 data_wo_o = train_df[(train_df<=5)&(train_df>=-5)].dropna(axis = 0)# 4
       clustering method
70
71 # after PCA
72 # create dataframe for features after pca
73 pca_df = pd.DataFrame(m_pca)
74 pca_df.columns = [ "{}th component".format(i+1) for i in range(11)]
75 pca_df.index = train_df.index
76
77 # outliers out of range of(-5,5)
78 # after pca ,samples outliers:
79 outliers_tickers = np.array(pca_df[(pca_df>5)|(pca_df<-5)].dropna(how =
        'all',axis = 0).index)
80 pca_o = pca_df.loc[outliers_tickers,:]
81
82 # after pca,samples without outliers
83 pca_wo_o = pca_df[(pca_df<=5)&(pca_df>=-5)].dropna(axis = 0)
```

Appendix B

Code: Model Implementation

B.1 Model: KMeans

```
1
2  from sklearn.cluster import KMeans
3  from sklearn.cluster import MiniBatchKMeans
4  from sklearn.cluster import Birch
5  from sklearn.metrics import silhouette_samples, silhouette_score
6  from sklearn.metrics import calinski_harabaz_score
7
8  # Evaluation method with sh score
9
10 def find_optimal_clusters1(n,data):
11     cluster_num = np.arange(2,n)
12     scores = []
13     for k in cluster_num:
14         kmean = KMeans(n_clusters = k,random_state = 42,algorithm='full
       ').fit(data)
15         labels = kmean.predict(data)
16         score = silhouette_score(data,labels)
17         scores.append(score)
18
19     print("Optimal number for clusters:{}".format(cluster_num[np.argmax
       (scores)]))
20     print("Optimal SH score:{}".format(max(scores)))
21     MS_cluster=plt.figure()
22     plt.plot(cluster_num,scores)
23     plt.legend(["Number of clusters","SH_score"],bbox_to_anchor=(1.1,1)
       )
24     plt.grid()
25     plt.title("Number of clusters and SH_score of Kmean")
26     plt.xlabel('Number of Clusters')
27     plt.ylabel('Score')
28     _ = plt.show()
29
30
31 # Modeling with After PCA Dataframe
32 find_optimal_clusters1(15,pca_wo_o) # max clustring range as 15
33
34 find_optimal_clusters1(15,data_wo_o)
35
36 # Elbow Method
37 def find_optimal_clusters2(n,data):
38     cluster_num = np.arange(2,n)
39     scores = []
40     for k in cluster_num:
41         kmean = KMeans(n_clusters = k, random_state = 42,algorithm='
       full').fit(data)
42         labels = kmean.predict(data)
```

```
        score = kmean.inertia_
        scores.append(score)


    MS_cluster=plt.figure()
    plt.plot(cluster_num,scores,'o-')
    plt.legend(["Number of clusters"],bbox_to_anchor=(1.1,1))
    plt.grid()
    plt.title("Number of clusters and Sum of square of Kmean")
    plt.xlabel('Number of Clusters')
    plt.ylabel('Score')
    _ = plt.show()

# With optimal parameters and after PCA dataframe.
find_optimal_clusters2(15,pca_wo_o)


# Gap statistic method
def find_optimal_clusters3(data, nrefs=3, maxClusters=15):
    """
    Calculates KMeans optimal K using Gap Statistic from Tibshirani, Walther, Hastie
    Params:
        data: ndarry of shape (n_samples, n_features)
        nrefs: number of sample reference datasets to create
        maxClusters: Maximum number of clusters to test for
    Returns: (gaps, optimalK)
    """
    gaps = np.zeros((len(range(1, maxClusters)),))
    resultsdf = pd.DataFrame({'clusterCount':[], 'gap':[]})
    for gap_index, k in enumerate(range(1, maxClusters)):

        # Holder for reference dispersion results
        refDisps = np.zeros(nrefs)

        # For n references, generate random sample and perform kmeans getting resulting dispersion of each loop
        for i in range(nrefs):

            # Create new random reference set
            randomReference = np.random.random_sample(size=data.shape)

            # Fit to it
            km = KMeans(k)
            km.fit(randomReference)

            refDisp = km.inertia_
            refDisps[i] = refDisp

        # Fit cluster to original data and create dispersion
        km = KMeans(k)
        km.fit(data)

        origDisp = km.inertia_
        origDisp = km.inertia_

        # Calculate gap statistic
        gap = np.log(np.mean(refDisps)) - np.log(origDisp)

        # Assign this loop's gap statistic to gaps
        gaps[gap_index] = gap

```

```
103         resultsdf = resultsdf.append({'clusterCount':k, 'gap':gap},
    ignore_index=True)
104
105     return (gaps.argmax() + 1, resultsdf)  # Plus 1 because index of 0
    means 1 cluster is optimal, index 2 = 3 clusters are optimal
106
107
108 # Use gap statistic to evaluate kmeans model
109 k, gapdf = find_optimal_clusters3(pca_wo_o, nrefs=5, maxClusters=15)
110 print ('Optimal k is: ', k)
111 plt.plot(gapdf.clusterCount, gapdf.gap, linewidth=3)
112 plt.scatter(gapdf[gapdf.clusterCount == k].clusterCount, gapdf[gapdf.
    clusterCount == k].gap, s=250, c='r')
113 plt.grid(True)
114 plt.xlabel('Cluster Count')
115 plt.ylabel('Gap Value')
116 plt.title('Gap Values by Cluster Count')
117 plt.show()
118
119 # Prediction distribution
120 plt.figure(figsize=(14, 6))
121 plt.subplot(121)
122 plt.hist(KMeans(8,random_state = 42,algorithm='full').fit_predict(
    pca_wo_o))
123 plt.title("Distribution with 8 Clustering group")
124
125 plt.subplot(122)
126 plt.hist(KMeans(11,random_state = 42,algorithm='full').fit_predict(
    pca_wo_o))
127 plt.title("Distribution with 11 Clustering group")
128
129
130 # outlier assignment
131 from scipy.spatial import distance
132 # center can be list or array
133 # assume each center correponds to label of [0,1,2...]
134 # return a list of label
135 def classify_outliers(centers,outliers_df):
136     labels = []
137     for sample in np.array(outliers_df):
138         min_center = float('inf')
139         min_center_th = -10086
140         for i,center in enumerate(centers):
141             dst = distance.euclidean(center,sample)
142             if dst<min_center:
143                 min_center = dst
144                 min_center_th = i
145         labels.append(min_center_th)
146     return labels
147
148 # example of using this fuction
149 # centers are calculated from the models, lets assume two random
    centers here
150 kmeans_11 = KMeans(11,random_state = 42,algorithm='full').fit(pca_wo_o)
151 centers = kmeans_11.cluster_centers_
152 outlier_labels = classify_outliers(centers,pca_o)
153 pca_wo_o_labels = kmeans_11.labels_
154 labels = np.append(pca_wo_o_labels,outlier_labels)
155
156 score = silhouette_score(pca_df,labels)
157 score2 = silhouette_score(pca_wo_o,pca_wo_o_labels)
158 print(score)
159 print(score2)
```

B.2 Model: MiniBatchKMeans

```
1
2  def find_optimal_clusters_mini(n,data,batch_size):
3      cluster_num = np.arange(2,n)
4      scores = []
5      for k in cluster_num:
6          minikmean = MiniBatchKMeans(n_clusters = k,random_state = 42,
       batch_size = batch_size).fit(data)
7          labels = minikmean.predict(data)
8          score = silhouette_score(data,labels)
9          scores.append(score)
10
11     print("Optimal number for clusters:{}".format(cluster_num[np.argmax
       (scores)]))
12     print("Optimal SH score:{}".format(max(scores)))
13     MS_cluster=plt.figure()
14     plt.plot(cluster_num,scores)
15     plt.legend(["Number of clusters","SH_score"],bbox_to_anchor=(1.1,1)
       )
16     plt.grid()
17     plt.title("Number of clusters and SH_score of MiniBatchKmean")
18     plt.xlabel('Number of Clusters')
19     plt.ylabel('CH Score')
20     _ = plt.show()
21
22 find_optimal_clusters_mini(15,pca_wo_o,10)
23
24
25 # Elbow Method
26 def find_optimal_clusters_mini2(n,data,batch_size):
27     cluster_num = np.arange(2,n)
28     scores = []
29     for k in cluster_num:
30         minikmean = MiniBatchKMeans(n_clusters = k,random_state = 42,
       batch_size = batch_size).fit(data)
31         labels = minikmean.predict(data)
32         score = minikmean.inertia_
33         scores.append(score)
34
35     MS_cluster=plt.figure()
36     plt.plot(cluster_num,scores,'o-')
37     plt.legend(["Number of clusters"],bbox_to_anchor=(1.1,1))
38     plt.grid()
39     plt.title("Number of clusters and Sum of square of MiniBatchKmean")
40     plt.xlabel('Number of Clusters')
41     plt.ylabel('Score')
42     _ = plt.show()
43
44
45 find_optimal_clusters_mini2(15,pca_wo_o,10)
46
47
48 # Gap statistic method
49 def find_optimal_clusters_mini3(data, nrefs=3, maxClusters=15):
50     """
51     Calculates KMeans optimal K using Gap Statistic from Tibshirani,
       Walther, Hastie
52     Params:
53         data: ndarry of shape (n_samples, n_features)
54         nrefs: number of sample reference datasets to create
55         maxClusters: Maximum number of clusters to test for
```

```
    Returns: (gaps, optimalK)
    """
    gaps = np.zeros((len(range(1, maxClusters)),))
    resultsdf = pd.DataFrame({'clusterCount':[], 'gap':[]})
    for gap_index, k in enumerate(range(1, maxClusters)):

        # Holder for reference dispersion results
        refDisps = np.zeros(nrefs)

        # For n references, generate random sample and perform kmeans
    getting resulting dispersion of each loop
        for i in range(nrefs):

            # Create new random reference set
            randomReference = np.random.random_sample(size=data.shape)

            # Fit to it
            km = MiniBatchKMeans(k)
            km.fit(randomReference)

            refDisp = km.inertia_
            refDisps[i] = refDisp

        # Fit cluster to original data and create dispersion
        km = MiniBatchKMeans(k)
        km.fit(data)

        origDisp = km.inertia_
        origDisp = km.inertia_

        # Calculate gap statistic
        gap = np.log(np.mean(refDisps)) - np.log(origDisp)

        # Assign this loop's gap statistic to gaps
        gaps[gap_index] = gap

        resultsdf = resultsdf.append({'clusterCount':k, 'gap':gap},
    ignore_index=True)

    return (gaps.argmax() + 1, resultsdf)


k, gapdf = find_optimal_clusters_mini3(pca_wo_o, nrefs=5, maxClusters
    =15)
print ('Optimal k is: ', k)

plt.plot(gapdf.clusterCount, gapdf.gap, linewidth=3)
plt.scatter(gapdf[gapdf.clusterCount == k].clusterCount, gapdf[gapdf.
    clusterCount == k].gap, s=250, c='r')
plt.grid(True)
plt.xlabel('Cluster Count')
plt.ylabel('Gap Value')
plt.title('Gap Values by Cluster Count')
plt.show()


# Prediction distribution

plt.figure(figsize=(14, 6))
plt.subplot(121)
plt.hist(MiniBatchKMeans(n_clusters = 6,random_state = 42, batch_size =
     10).fit_predict(pca_wo_o))
plt.title("Distribution with 8 Clustering group")
```

```
114
115 plt.subplot(122)
116 plt.hist(MiniBatchKMeans(n_clusters = 11,random_state = 42, batch_size
        = 10).fit_predict(pca_wo_o))
117 plt.title("Distribution with 11 Clustering group")
118
119
120 # outlier assignment
121
122 #example of using this fuction
123 #centers are calculated from the models, lets assume two random centers
        here
124 MiniBatchKMeans_6 = MiniBatchKMeans(n_clusters = 6,random_state = 42,
        batch_size = 10).fit(pca_wo_o)
125 minicenters = MiniBatchKMeans_6.cluster_centers_
126 minioutlier_labels = classify_outliers(minicenters,pca_o)
127 minipca_wo_o_labels = MiniBatchKMeans_6.labels_
128 minilabels = np.append(minipca_wo_o_labels,minioutlier_labels)
129
130 miniscore = silhouette_score(pca_df,minilabels)
131 miniscore2 = silhouette_score(pca_wo_o,minipca_wo_o_labels)
132 print(miniscore)
133 print(miniscore2)
```

B.3 Model: Birch

```

# Grid search
def find_optimal_clusters_birch(data,bf,thd):
    cluster_num = [2,3,4,5,6,7,8,9,10,11,12,13,14]
    scores = []
    for n in cluster_num:
        brc = Birch(branching_factor=bf, n_clusters=n, threshold=thd,
    compute_labels=True).fit(data)
        label = brc.predict(data)
        score = calinski_harabaz_score(data,label)
        scores.append(score)
    print("Optimal number for clusters:{}".format(cluster_num[np.argmax
    (scores)]))
    print("Optimal CH score:{}".format(max(scores)))

    MS_cluster=plt.figure()
    plt.plot(cluster_num,scores)
    plt.legend(["Number of clusters","CH_score"],bbox_to_anchor=(1.1,1)
    )
    plt.grid()
    plt.title("Number of clusters and CH_score of Birch")
    plt.xlabel('Number of Clusters')
    plt.ylabel('CH Score')
    _ = plt.show()


def grid_search_birch(data):
    threshold= [0.1,0.2,0.3,0.5,0.7,1]
    branching_factor = [20, 30, 40, 50, 60, 80, 100]
    for i in threshold:
        for j in branching_factor:
            try:
                find_optimal_clusters_birch(data,j,i)
                print(j,i)
            except:
                print('not appliable')

grid_search_birch(pca_wo_o)

# find optimal cluster number
find_optimal_clusters_birch(pca_wo_o, 50 ,1)

# prediction distribution
bir = Birch(branching_factor=50, n_clusters=4, threshold=1,
    compute_labels=True).fit(pca_wo_o)
bir2 = Birch(branching_factor=50, n_clusters=8, threshold=1,
    compute_labels=True).fit(pca_wo_o)

plt.figure(figsize = (14,6))
plt.subplot(121)
plt.hist(bir.predict(pca_wo_o))
plt.title("Distribution with 4 Clustering group")

plt.subplot(122)
plt.hist(bir2.predict(pca_wo_o))
plt.title("Distribution with 8 Clustering group")

# outlier assignment
#example of using this fuction
```

```
55 #centers are calculated from the models, lets assume two random centers
       here
56
57 bircenters = bir.subcluster_centers_
58 biroutlier_labels = classify_outliers(minicenters,pca_o)
59 birpca_wo_o_labels = bir.labels_
60 birlabels = np.append(birpca_wo_o_labels,biroutlier_labels)
61
62 birscore = silhouette_score(pca_df,birlabels)
63 birscore2 = silhouette_score(pca_wo_o,birpca_wo_o_labels)
64 print(birscore)
65 print(birscore2)
```

B.4 Model: Agglomerative

```

from scipy.cluster.hierarchy import dendrogram, linkage,
    set_link_color_palette
from scipy.cluster.hierarchy import cophenet
from scipy.spatial.distance import pdist
import pylab

from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import calinski_harabaz_score, silhouette_score

# define grid search function for Agglomerative Clustering

def AC_grid_search(df,n,link_list,aff_list):
    num_of_cluster = np.arange(2,n+1,1)
    ch_score = []
    sh_score = []
    real_cluster = []
    for k in num_of_cluster:
        opt_ch = 0
        opt_sh = 0
        opt_link = 'null'
        opt_aff = 'null'
        n_cluster = 0
        for i in link_list:
            for j in aff_list:
                try:
                    AC_model = AgglomerativeClustering(n_clusters=k,
    linkage = i, affinity=j)
                    predicted = AC_model.fit_predict(df)
                    temp_score = calinski_harabaz_score(X = df,labels =
     predicted )
                    if temp_score > opt_ch:
                        opt_ch = temp_score
                        opt_sh = silhouette_score(X = df, labels =
    predicted)*100
                        opt_link = i
                        opt_aff = j
                        n_cluster = len(set(predicted))
                except:
                    pass
        print('For',k, 'clusters, Optimal combination is: linkage = ',
    opt_link, 'affinity = ', opt_aff)
        print('    Highest CH_score is: ',opt_ch, 'Corresponding
    SH_score is: ', opt_sh)
        ch_score.append(opt_ch)
        sh_score.append(opt_sh)
        real_cluster.append(n_cluster)

    AC_clusters=plt.figure()
    plt.plot(num_of_cluster,ch_score)
    plt.plot(num_of_cluster,sh_score)
    plt.plot(num_of_cluster, real_cluster)
    plt.grid()
    plt.legend(["ch_score", "sh_score * 100","real_cluster_num"],
    bbox_to_anchor=(1.1,1))
    plt.title("CH_score with different number of clusters")
    plt.xlabel('Number of Clusters')
    _ = plt.show()

# parameter list
```

```
affinity_list = ['euclidean','l1','l2','manhattan','cosine','precomputed']
linkage_list = ['ward','complete', 'average', 'single']

# before PCA
AC_grid_search(train_df,20,linkage_list,affinity_list)

# Plot dendrogram
ward_nopca = linkage(train_df, 'ward')
plt.figure(figsize=(25, 10))
labelsize=20
ticksize=15
set_link_color_palette(['firebrick','lightcoral','darkorange','gold','yellowgreen','mediumaquamarine','lightseagreen','skyblue','steelblue','slateblue','rebeccapurple','darkmagenta'])
plt.title('Agglomerative Clustering Dendrogram (ward, no PCA)', fontsize=labelsize)
plt.xlabel('stock', fontsize=labelsize)
plt.ylabel('distance', fontsize=labelsize)
dendrogram(
    ward_nopca,
    color_threshold = 28,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
pylab.yticks(fontsize=ticksize)
pylab.xticks(rotation=-90, fontsize=ticksize)
plt.show()

# plot histgram for optimal result
ac_nopca2 = AgglomerativeClustering(n_clusters=12, linkage =  'ward')
ac_nopca_predicted2 = ac_nopca2.fit_predict(train_df)
print('ch_score = %.2f' % calinski_harabaz_score(X = train_df,labels = ac_nopca_predicted2 ))
print('sh-score = %.2f' % silhouette_score(X = train_df, labels = ac_nopca_predicted2))
plt.hist(ac_nopca_predicted2,bins=12)

# calculate std for clustering result
result = pd.DataFrame(ac_nopca_predicted2)
result[0].value_counts().std()


# after PCA, with outliers
AC_grid_search(m_pca,20,linkage_list,affinity_list)

# Dendrogram
ward_pca = linkage(m_pca, 'ward')
plt.figure(figsize=(25, 10))
labelsize=20
ticksize=15
plt.title('Agglomerative Clustering Dendrogram (ward, PCA, with outliers)', fontsize=labelsize)
plt.xlabel('stock', fontsize=labelsize)
plt.ylabel('distance', fontsize=labelsize)
dendrogram(
    ward_pca,
    color_threshold = 28,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
pylab.yticks(fontsize=ticksize)
pylab.xticks(rotation=-90, fontsize=ticksize)
```

```
plt.show()

# histogram
ac_pca1 = AgglomerativeClustering(n_clusters=9, linkage =  'ward')
ac_pca_predicted1 = ac_pca1.fit_predict(m_pca)
print('ch_score = %.2f' % calinski_harabaz_score(X = m_pca,labels =
    ac_pca_predicted1 ))
print('sh-score = %.2f' % silhouette_score(X = m_pca, labels =
    ac_pca_predicted1))
plt.hist(ac_pca_predicted1,bins=9)

# clustering result std
result = pd.DataFrame(ac_pca_predicted1)
result[0].value_counts().std()


# after PCA without outliers
AC_grid_search(pca_wo_o,20,linkage_list,affinity_list)

# Dendrogram
ward_pca = linkage(pca_wo_o, 'ward')
plt.figure(figsize=(25, 10))
labelsize=20
ticksize=15
plt.title('Agglomerative Clustering Dendrogram (ward, PCA, without
    outliers)', fontsize=labelsize)
plt.xlabel('stock', fontsize=labelsize)
plt.ylabel('distance', fontsize=labelsize)
dendrogram(
    ward_pca,
    color_threshold = 21.5,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
pylab.yticks(fontsize=ticksize)
pylab.xticks(rotation=-90, fontsize=ticksize)
plt.show()

# histogram
ac_wo_o1 = AgglomerativeClustering(n_clusters=6, linkage =  'ward')
ac_wo_o_predicted1 = ac_wo_o1.fit_predict(pca_wo_o)
print('ch_score = %.2f' % calinski_harabaz_score(X = pca_wo_o,labels =
    ac_wo_o_predicted1 ))
print('sh-score = %.2f' % silhouette_score(X = pca_wo_o, labels =
    ac_wo_o_predicted1))
plt.hist(ac_wo_o_predicted1,bins=6)

# result std
result = pd.DataFrame(ac_wo_o_predicted1)
result[0].value_counts().std()

# dendrogram for n = 13
ward_pca = linkage(pca_wo_o, 'ward')
plt.figure(figsize=(25, 10))
labelsize=20
ticksize=15
set_link_color_palette(['firebrick','lightcoral','darkorange','gold','
    yellowgreen','limegreen','mediumaquamarine','lightseagreen','
    skyblue','steelblue','slateblue','rebeccapurple','darkmagenta'])
plt.title('Agglomerative Clustering Dendrogram (ward, PCA, without
    outliers)', fontsize=labelsize)
plt.xlabel('stock', fontsize=labelsize)
plt.ylabel('distance', fontsize=labelsize)
```

```
dendrogram(
    ward_pca,
    color_threshold = 12.4,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
pylab.yticks(fontsize=ticksize)
pylab.xticks(rotation=-90, fontsize=ticksize)
plt.show()

# histogram for n = 13
ac_wo_o2 = AgglomerativeClustering(n_clusters=13, linkage =  'ward')
ac_wo_o_predicted2 = ac_wo_o2.fit_predict(pca_wo_o)
print('ch_score = %.2f' % calinski_harabaz_score(X = pca_wo_o,labels =
    ac_wo_o_predicted2 ))
print('sh-score = %.2f' % silhouette_score(X = pca_wo_o, labels =
    ac_wo_o_predicted2))
plt.hist(ac_wo_o_predicted2,bins=13)

# result std
result = pd.DataFrame(ac_wo_o_predicted2)
result[0].value_counts().std()

# labelling outliers
# define label function
from scipy.spatial import distance
def classify_outliers(center_df, outliers_df):
    labels = []
    for sample in np.array(outliers_df):
        min_center = float('inf')
        min_center_th = -10086
        i=0
        for center in np.array(center_df):
            dst = distance.euclidean(center,sample)
            if dst<min_center:
                min_center = dst
                min_center_th = i
                i=i+1
        labels.append(min_center_th)
    return labels

pca_wo_o1 = pca_wo_o.copy()
pca_wo_o1['label'] = ac_wo_o_predicted1

# calculate center for n = 6
ac_center1 = pd.DataFrame()
for i in range(6):
    ac_center1[i] = np.mean(pca_wo_o1[pca_wo_o1['label']==i],axis=0)

ac_center1 = ac_center1.drop(['label']).transpose()

# plot histogram after labelling outliers
ac_o_labels1 = classify_outliers(ac_center1,pca_o)
print( calinski_harabaz_score(X =pd.concat([pca_wo_o,pca_o])  ,labels =
     list(ac_wo_o_predicted1)+ac_o_labels1 ))
plt.hist(list(ac_wo_o_predicted1)+ac_o_labels1,bins=6)

# result std
result = pd.DataFrame(ac_wo_o_predicted1)
result[0].value_counts().std()

# get clustering center for n =13
pca_wo_o2 = pca_wo_o.copy()
```

```
225 pca_wo_o2['label'] = ac_wo_o_predicted2
226
227 ac_center2 = pd.DataFrame()
228 for i in range(13):
229     ac_center2[i] = np.mean(pca_wo_o2[pca_wo_o2['label']==i],axis=0)
230
231 ac_center2 = ac_center2.drop(['label']).transpose()
232
233 # histogram
234 ac_o_labels2 = classify_outliers(ac_center2,pca_o)
235 print( calinski_harabaz_score(X =pd.concat([pca_wo_o,pca_o])  ,labels =
        list(ac_wo_o_predicted2)+ac_o_labels2 ))
236 plt.hist(list(ac_wo_o_predicted2)+ac_o_labels2,bins=13)
237
238 # result std
239 result = pd.DataFrame(ac_wo_o_predicted2)
240 result[0].value_counts().std()
```

B.5 Model: Spectral

```
from sklearn.cluster import SpectralClustering

# define grid search for spectral clustering
def SC_grid_search(df,n,ninit,gamma,aff_list,neighbors):
    num_of_cluster = np.arange(2,n+1,1)
    ch_score = []
    sh_score = []
    real_cluster = []
    for k in num_of_cluster:
        opt_ch = 0
        opt_sh = 0
        opt_init = 0
        opt_gamma = 0
        opt_aff = 'null'
        opt_nei = 0
        n_cluster = 0
        for i in ninit:
            for j in gamma:
                for l in aff_list:
                    for m in neighbors:
                        try:
                            SC_model = SpectralClustering(n_clusters=k,
     n_init= i, gamma= j, affinity=l, n_neighbors=m, random_state=42)
                            predicted = SC_model.fit_predict(df)
                            temp_score = calinski_harabaz_score(X = df,
    labels = predicted )
                            if temp_score > opt_ch:
                                opt_ch = temp_score
                                opt_sh = silhouette_score(X = df,
    labels = predicted)*100
                                opt_init = i
                                opt_gamma = j
                                opt_aff = l
                                opt_nei = m
                                n_cluster = len(set(predicted))
                        except:
                            pass
        print('For',k, 'clusters, Optimal combination is: n_init = ',
    opt_init, ', gamma = ', opt_gamma,
              ',affinity = ', opt_aff, ',number of neighbors = ',
    opt_nei)
        print('    Highest CH_score is: ',opt_ch, 'Corresponding
    SH_score is: ', opt_sh)
        ch_score.append(opt_ch)
        sh_score.append(opt_sh)
        real_cluster.append(n_cluster)

    AC_clusters=plt.figure()
    plt.plot(num_of_cluster,ch_score)
    plt.plot(num_of_cluster,sh_score)
    plt.plot(num_of_cluster, real_cluster)
    plt.grid()
    plt.legend(["ch_score", "sh_score * 100","real_cluster_num"],
    bbox_to_anchor=(1.1,1))
    plt.title("CH_score with different number of clusters")
    plt.xlabel('Number of Clusters')
    _ = plt.show()

# parameter matrix
init_list = np.arange(5,30,5)
```

```
gamma_list = np.arange(0.6,1.5,0.2)
aff_list = ['nearest_neighbors','precomputed','rbf']
neighbor_list = np.arange(5,30,5)

import warnings
warnings.filterwarnings('ignore')

# before pca
SC_grid_search(train_df,20,init_list,gamma_list,aff_list,neighbor_list)

# histogram
sc_model_nopca = SpectralClustering(n_clusters=4, n_init= 5, gamma =
    0.6, affinity =  'nearest_neighbors', n_neighbors =  10,
    random_state=42)
sc_predicted_nopca = sc_model_nopca.fit_predict(train_df)
print('ch_score = %.2f' % calinski_harabaz_score(X = train_df,labels =
    sc_predicted_nopca ))
print('sh-score = %.2f' % silhouette_score(X = train_df, labels =
    sc_predicted_nopca))
plt.hist(sc_predicted_nopca,bins=4)

# result std
result = pd.DataFrame(sc_predicted_nopca)
result[0].value_counts().std()

# histogram
sc_model_nopca = SpectralClustering(n_clusters=9, n_init= 10, gamma =
    0.6, affinity =  'nearest_neighbors', n_neighbors =  10,
    random_state=42)
sc_predicted_nopca = sc_model_nopca.fit_predict(train_df)
print('ch_score = %.2f' % calinski_harabaz_score(X = train_df,labels =
    sc_predicted_nopca ))
print('sh-score = %.2f' % silhouette_score(X = train_df, labels =
    sc_predicted_nopca))
plt.hist(sc_predicted_nopca, bins =9)

# result std
result = pd.DataFrame(sc_predicted_nopca)
result[0].value_counts().std()

# after std with outliers
SC_grid_search(m_pca,20,init_list,gamma_list,aff_list,neighbor_list)

# histogram
sc_model_pca = SpectralClustering(n_clusters=4, n_init= 5, gamma = 0.6,
     affinity =  'nearest_neighbors', n_neighbors =  10, random_state
    =42)
sc_predicted_pca = sc_model_pca.fit_predict(m_pca)
print('ch_score = %.2f' % calinski_harabaz_score(X = m_pca,labels =
    sc_predicted_pca ))
print('sh-score = %.2f' % silhouette_score(X = m_pca, labels =
    sc_predicted_pca))
plt.hist(sc_predicted_pca,bins=4)

# result std
result = pd.DataFrame(sc_predicted_pca)
result[0].value_counts().std()

# histogram
sc_model_pca = SpectralClustering(n_clusters=8, n_init= 20, gamma =
    0.6, affinity =  'nearest_neighbors', n_neighbors =  15,
    random_state=42)
sc_predicted_pca = sc_model_pca.fit_predict(m_pca)
```

```
print('ch_score = %.2f' % calinski_harabaz_score(X = m_pca,labels =
    sc_predicted_pca ))
print('sh-score = %.2f' % silhouette_score(X = m_pca, labels =
    sc_predicted_pca))
plt.hist(sc_predicted_pca,bins=8)

# result std
result = pd.DataFrame(sc_predicted_pca)
result[0].value_counts().std()

# histogram
sc_model_pca = SpectralClustering(n_clusters=12, n_init= 10, gamma =
    0.6, affinity =  'nearest_neighbors', n_neighbors =  15,
    random_state=42)
sc_predicted_pca = sc_model_pca.fit_predict(m_pca)
print('ch_score = %.2f' % calinski_harabaz_score(X = m_pca,labels =
    sc_predicted_pca ))
print('sh-score = %.2f' % silhouette_score(X = m_pca, labels =
    sc_predicted_pca))
plt.hist(sc_predicted_pca,bins=12)

# result std
result = pd.DataFrame(sc_predicted_pca)
result[0].value_counts().std()

# after pca without outliers
SC_grid_search(pca_wo_o,20,init_list,gamma_list,aff_list,neighbor_list)

# histogram
sc_model_wo_o = SpectralClustering(n_clusters=8, n_init= 5, gamma =
    0.60, affinity =  'nearest_neighbors', n_neighbors =  25,
    random_state=42)
sc_predicted_wo_o = sc_model_wo_o.fit_predict(pca_wo_o)
print('ch_score = %.2f' % calinski_harabaz_score(X = pca_wo_o,labels =
    sc_predicted_wo_o ))
print('sh-score = %.2f' % silhouette_score(X = pca_wo_o, labels =
    sc_predicted_wo_o))
plt.hist(sc_predicted_wo_o,bins=8)

# result std
result = pd.DataFrame(sc_predicted_wo_o)
result[0].value_counts().std()

# labelling outliers
pca_wo_o3 = pca_wo_o.copy()
pca_wo_o3['label'] = sc_predicted_wo_o

sc_center = pd.DataFrame()
for i in range(8):
    sc_center[i] = np.mean(pca_wo_o3[pca_wo_o3['label']==i],axis=0)

sc_center = sc_center.drop(['label']).transpose()

sc_o_labels = classify_outliers(sc_center,pca_o)
print( calinski_harabaz_score(X =pd.concat([pca_wo_o,pca_o])  ,labels =
     list(sc_predicted_wo_o)+sc_o_labels ))
plt.hist(list(sc_predicted_wo_o)+sc_o_labels,bins=8)
```

B.6 Model: Mean Shift

```

from sklearn.cluster import MeanShift
import sklearn.metrics
from sklearn.metrics import silhouette_score
from sklearn.metrics import davies_bouldin_score

MS_cluster = MeanShift(bandwidth=3.3,bin_seeding = True,cluster_all=
    True)
predicted = MS_cluster.fit_predict(train_df )

from collections import Counter
result = Counter(predicted)
print(result)
plt.hist(predicted)

MS_cluster = MeanShift(bandwidth=3.3,bin_seeding = True,cluster_all=
    True)
predicted = MS_cluster.fit_predict(data_wo_o )

from collections import Counter
result = Counter(predicted)
print(result)

# before pca with outliers
def find_optimal_bandwidth(start,end,interval):
    num_of_cluster = []
    ch_score = []
    s_score = []
    x_axis = []
    for i in np.arange(start,end,interval):
        try:
            MS_cluster = MeanShift(bandwidth=i,bin_seeding = True,
    cluster_all=True)
            predicted = MS_cluster.fit_predict(train_df )
            ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
    train_df ,labels = predicted ))
            s_score.append(100*silhouette_score(X = train_df ,labels =
    predicted))
        except:
            continue

        x_axis.append(i)
        num_of_cluster.append(max(predicted)+1)
    MS_cluster_bandwidth=plt.figure()
    plt.plot(x_axis,num_of_cluster)
    plt.plot(x_axis,ch_score)
    plt.plot(x_axis,s_score)
    plt.legend(["Number of clusters","CH_score","100 * Silhouette_score
    "],bbox_to_anchor=(1.1,1))
    plt.grid()
    plt.title("Number of clusters silhouette_score and CH_score with
    different bandwidth")
    plt.xlabel('Bandwidth')
    plt.ylabel('Number of Clusters/Score')
    _ = plt.show()

    print("Optimal Bandwidth:")
    print(np.arange(start,end,interval)[np.argmax(ch_score)])
    print("Optimal number for clusters:")
    print(num_of_cluster[np.argmax(ch_score)])
```

```
    print("Largest CH_score:")
    print(max(ch_score))

    return [ch_score,num_of_cluster]

ch_score_ms_2,num_of_cluster_ms_2 = find_optimal_bandwidth(1.5,4.5,0.1)

from collections import Counter
MS_cluster = MeanShift(bandwidth=2.86,bin_seeding = True,cluster_all=
    True)
predicted = MS_cluster.fit_predict(train_df )

result = Counter(predicted)
print(result)
plt.hist(predicted)

# before pca without outliers
def find_optimal_bandwidth(start,end,interval):
    num_of_cluster = []
    ch_score = []
    s_score = []
    x_axis = []
    for i in np.arange(start,end,interval):
        try:
            MS_cluster = MeanShift(bandwidth=i,bin_seeding = True,
    cluster_all=True)
            predicted = MS_cluster.fit_predict(data_wo_o )
            ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
    data_wo_o ,labels = predicted ))
            s_score.append(100*silhouette_score(X = data_wo_o ,labels =
     predicted))
        except:
            continue

        x_axis.append(i)
        num_of_cluster.append(max(predicted)+1)
    MS_cluster_bandwidth=plt.figure()
    plt.plot(x_axis,num_of_cluster)
    plt.plot(x_axis,ch_score)
    plt.plot(x_axis,s_score)
    plt.legend(["Number of clusters","CH_score","100 * Silhouette_score
    "],bbox_to_anchor=(1.1,1))
    plt.grid()
    plt.title("Number of clusters silhouette_score and CH_score with
    different bandwidth")
    plt.xlabel('Bandwidth')
    plt.ylabel('Number of Clusters/Score')
    _ = plt.show()

    print("Optimal Bandwidth:")
    print(np.arange(start,end,interval)[np.argmax(ch_score)])
    print("Optimal number for clusters:")
    print(num_of_cluster[np.argmax(ch_score)])
    print("Largest CH_score:")
    print(max(ch_score))

    return [ch_score,num_of_cluster]

ch_score_ms_2,num_of_cluster_ms_2 = find_optimal_bandwidth
    (1.5,4.5,0.01)

MS_cluster = MeanShift(bandwidth=2.75,bin_seeding = True,cluster_all=
    True)
```

```
predicted = MS_cluster.fit_predict(data_wo_o )

result = Counter(predicted)
print(result)
plt.hist(predicted)

# after pca with outliers
def find_optimal_bandwidth(start,end,interval):
    num_of_cluster = []
    ch_score = []
    s_score = []
    x_axis = []
    for i in np.arange(start,end,interval):
        try:
            MS_cluster = MeanShift(bandwidth=i,bin_seeding = True,
    cluster_all=True)
            predicted = MS_cluster.fit_predict(pca_df )
            ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
    pca_df ,labels = predicted ))
            s_score.append(100*silhouette_score(X = pca_df ,labels =
    predicted))
        except:
            continue

        x_axis.append(i)
        num_of_cluster.append(max(predicted)+1)
    MS_cluster_bandwidth=plt.figure()
    plt.plot(x_axis,num_of_cluster)
    plt.plot(x_axis,ch_score)
    plt.plot(x_axis,s_score)
    plt.legend(["Number of clusters","CH_score","100 * Silhouette_score
    "],bbox_to_anchor=(1.1,1))
    plt.grid()
    plt.title("Number of clusters silhouette_score and CH_score with
    different bandwidth")
    plt.xlabel('Bandwidth')
    plt.ylabel('Number of Clusters/Score')
    _ = plt.show()

    print("Optimal Bandwidth:")
    print(np.arange(start,end,interval)[np.argmax(ch_score)])
    print("Optimal number for clusters:")
    print(num_of_cluster[np.argmax(ch_score)])
    print("Largest CH_score:")
    print(max(ch_score))

    return [ch_score,num_of_cluster]

ch_score_ms_2,num_of_cluster_ms_2 = find_optimal_bandwidth(3,10,0.1)

MS_cluster = MeanShift(bandwidth=5,bin_seeding = True,cluster_all=True)
predicted = MS_cluster.fit_predict(pca_df )

result = Counter(predicted)
print(result)
plt.hist(predicted)

# after pca without outliers
def find_optimal_bandwidth(start,end,interval):
    num_of_cluster = []
    ch_score = []
    s_score = []
    x_axis = []
```

```
    for i in np.arange(start,end,interval):
        try:
            MS_cluster = MeanShift(bandwidth=i,bin_seeding = True,
    cluster_all=True)
            predicted = MS_cluster.fit_predict(pca_wo_o )
            ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
    pca_wo_o ,labels = predicted ))
            s_score.append(100*silhouette_score(X = pca_wo_o ,labels =
    predicted))
        except:
            continue

        x_axis.append(i)
        num_of_cluster.append(max(predicted)+1)
    MS_cluster_bandwidth=plt.figure()
    plt.plot(x_axis,num_of_cluster)
    plt.plot(x_axis,ch_score)
    plt.plot(x_axis,s_score)
    plt.legend(["Number of clusters","CH_score","100 * Silhouette_score
    "],bbox_to_anchor=(1.1,1))
    plt.grid()
    plt.title("Number of clusters silhouette_score and CH_score with
    different bandwidth")
    plt.xlabel('Bandwidth')
    plt.ylabel('Number of Clusters/Score')
    _ = plt.show()

    print("Optimal Bandwidth:")
    print(np.arange(start,end,interval)[np.argmax(ch_score)])
    print("Optimal number for clusters:")
    print(num_of_cluster[np.argmax(ch_score)])
    print("Largest CH_score:")
    print(max(ch_score))

    return [ch_score,num_of_cluster]

ch_score_ms_2,num_of_cluster_ms_2 = find_optimal_bandwidth(3,12,0.1)
```

B.7 Model: Affinity Propagation

```
from sklearn.cluster import AffinityPropagation

# before pca with outliers
def find_optimal_damping(start,end,interval):
    num_of_cluster = []
    ch_score = []
    s_score = []
    x = []
    for i in np.arange(start,end,interval):
        AP_cluster = AffinityPropagation(preference=i,convergence_iter
    = 50,max_iter=400)
        predicted = AP_cluster.fit_predict(pca_df )
        try:
            ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
    pca_df ,labels = predicted ))
        except:
            continue
        x.append(i)
        num_of_cluster.append(len(set(predicted) ))

        s_score.append(100*silhouette_score(X = pca_df ,labels =
    predicted))
    AP_cluster_damping=plt.figure()
    plt.plot(x,num_of_cluster)
    plt.plot(x,ch_score)
    plt.plot(x,s_score)

    plt.legend(["Number of clusters","CH_score","Silhouette_score"],
    bbox_to_anchor=(1.1,1))
    plt.grid()
    plt.title("Number of clusters,Silhouette_score and CH_score with
    different performance")
    plt.xlabel('damping')
    plt.ylabel('Number of Clusters')
    _ = plt.show()

    print("Optimal preference:")
    print(x[np.argmax(ch_score)])
    print("Optimal number for clusters:")
    print(num_of_cluster[np.argmax(ch_score)])
    print("Largest CH_score:")
    print(max(ch_score))

    return [ch_score,num_of_cluster]

ch_score_ms,num_of_cluster_ms = find_optimal_damping(-600,-60,10)

for item in [-500,-450,-400,-200]:
    MS_cluster = AffinityPropagation(preference = item,convergence_iter
    = 50,max_iter=400)
    predicted =MS_cluster.fit_predict(train_df)
    print("Number of Clusters:{}".format(max(predicted)+1))

    from collections import Counter
    result = Counter(predicted)
    plt.figure()
    plt.hist(predicted,bins= range(max(predicted)+1))
    plt.show()

# before pca without outliers
```

```
55 def find_optimal_damping(start,end,interval):
56     num_of_cluster = []
57     ch_score = []
58     s_score = []
59     x = []
60     for i in np.arange(start,end,interval):
61         AP_cluster = AffinityPropagation(preference=i,convergence_iter
       = 50,max_iter=400)
62         predicted = AP_cluster.fit_predict(data_wo_o )
63         try:
64             ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
       data_wo_o ,labels = predicted ))
65         except:
66             continue
67         x.append(i)
68         num_of_cluster.append(len(set(predicted) ))
69 
70         s_score.append(100*silhouette_score(X = data_wo_o ,labels =
       predicted))
71     AP_cluster_damping=plt.figure()
72     plt.plot(x,num_of_cluster)
73     plt.plot(x,ch_score)
74     plt.plot(x,s_score)
75 
76     plt.legend(["Number of clusters","CH_score","Silhouette_score"],
       bbox_to_anchor=(1.1,1))
77     plt.grid()
78     plt.title("Number of clusters,Silhouette_score and CH_score with
       different performance")
79     plt.xlabel('damping')
80     plt.ylabel('Number of Clusters')
81     _ = plt.show()
82 
83     print("Optimal performance:")
84     print(x[np.argmax(ch_score)])
85     print("Optimal number for clusters:")
86     print(num_of_cluster[np.argmax(ch_score)])
87     print("Largest CH_score:")
88     print(max(ch_score))
89 
90     return [ch_score,num_of_cluster]
91 
92 ch_score_ms,num_of_cluster_ms = find_optimal_damping(-600,-60,10)
93 
94 for item in [-300,-240,-150,-100]:
95     MS_cluster = AffinityPropagation(preference = item,convergence_iter
       = 50,max_iter=400)
96     predicted =MS_cluster.fit_predict(data_wo_o)
97     print("Number of Clusters:{}".format(max(predicted)+1))
98 
99     from collections import Counter
100    result = Counter(predicted)
101    plt.figure()
102    plt.hist(predicted,bins= range(max(predicted)+1))
103    plt.show()
104 
105 AP_cluster = AffinityPropagation(preference = -150,convergence_iter =
       50,max_iter=400)
106 predicted = AP_cluster.fit_predict(data_wo_o)
107 print("Number of Clusters:{}".format(max(predicted)))
108 
109 from collections import Counter
110 result = Counter(predicted)
```

```
111
112 plt.figure()
113 plt.hist(predicted,bins= range(max(predicted)+1))
114 plt.show()
115 display(sorted(result.items(), key=lambda x: x[0]))
116 print("CH score = {}".format(sklearn.metrics.calinski_harabaz_score(X =
      data_wo_o ,labels = predicted )))
117
118 # outliers
119 from scipy.spatial import distance
120 #center can be list or array
121 #assume each center correponds to label of [0,1,2...]
122 #return a list of label
123 def classify_outliers(centers,outliers_df):
124     labels = []
125     for sample in np.array(outliers_df):
126         min_center = float('inf')
127         min_center_th = -10086
128         for i,center in enumerate(centers):
129             dst = distance.euclidean(center,sample)
130             if dst<min_center:
131                 min_center = dst
132                 min_center_th = i
133         labels.append(min_center_th)
134     return labels
135
136 out_labels = classify_outliers(AP_cluster.cluster_centers_,data_o)
137
138 sklearn.metrics.calinski_harabaz_score(X =pd.concat([data_wo_o,data_o])
      ,labels = list(predicted)+out_labels )
139
140 # after pca with outliers
141 def find_optimal_damping(start,end,interval):
142     num_of_cluster = []
143     ch_score = []
144     s_score = []
145     x = []
146     for i in np.arange(start,end,interval):
147         AP_cluster = AffinityPropagation(preference=i,convergence_iter
    = 50,max_iter=400)
148         predicted = AP_cluster.fit_predict(pca_df )
149         try:
150             ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
    pca_df ,labels = predicted ))
151         except:
152             continue
153         x.append(i)
154         num_of_cluster.append(len(set(predicted) ))
155
156         s_score.append(100*silhouette_score(X = pca_df ,labels =
    predicted))
157     AP_cluster_damping=plt.figure()
158     plt.plot(x,num_of_cluster)
159     plt.plot(x,ch_score)
160     plt.plot(x,s_score)
161
162     plt.legend(["Number of clusters","CH_score","Silhouette_score"],
    bbox_to_anchor=(1.1,1))
163     plt.grid()
164     plt.title("Number of clusters,Silhouette_score and CH_score with
    different performance")
165     plt.xlabel('damping')
166     plt.ylabel('Number of Clusters')
```

```
    _ = plt.show()

    print("Optimal performance:")
    print(x[np.argmax(ch_score)])
    print("Optimal number for clusters:")
    print(num_of_cluster[np.argmax(ch_score)])
    print("Largest CH_score:")
    print(max(ch_score))

    return [ch_score,num_of_cluster]

ch_score_ms,num_of_cluster_ms = find_optimal_damping(-400,-60,10)

AP_cluster = AffinityPropagation(preference = -300,convergence_iter =
    50,max_iter=400)
predicted = AP_cluster.fit_predict(pca_df)
print("Number of Clusters:{}".format(max(predicted)))

from collections import Counter
result = Counter(predicted)

plt.figure()
plt.hist(predicted,bins= max(predicted)+1)
plt.show()
display(sorted(result.items(), key=lambda x: x[0]))

# after pca without outliers
def find_optimal_damping(start,end,interval):
    num_of_cluster = []
    ch_score = []
    s_score = []
    x = []
    for i in np.arange(start,end,interval):
        AP_cluster = AffinityPropagation(preference=i,convergence_iter
    = 50,max_iter=400)
        predicted = AP_cluster.fit_predict(pca_wo_o )
        try:
            ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
    pca_wo_o ,labels = predicted ))
        except:
            continue
        x.append(i)
        num_of_cluster.append(len(set(predicted) ))

        s_score.append(100*silhouette_score(X = pca_wo_o ,labels =
    predicted))
    AP_cluster_damping=plt.figure()
    plt.plot(x,num_of_cluster)
    plt.plot(x,ch_score)
    plt.plot(x,s_score)

    plt.legend(["Number of clusters","CH_score","Silhouette_score"],
    bbox_to_anchor=(1.1,1))
    plt.grid()
    plt.title("Number of clusters,Silhouette_score and CH_score with
    different performance")
    plt.xlabel('damping')
    plt.ylabel('Number of Clusters')
    _ = plt.show()

    print("Optimal performance:")
    print(x[np.argmax(ch_score)])
    print("Optimal number for clusters:")
```

```
224     print(num_of_cluster[np.argmax(ch_score)])
225     print("Largest CH_score:")
226     print(max(ch_score))
227 
228     return [ch_score,num_of_cluster]
229 
230 ch_score_ms,num_of_cluster_ms = find_optimal_damping(-400,-60,10)
231 
232 AP_cluster = AffinityPropagation(preference = -125,convergence_iter =
        50,max_iter=400)
233 predicted = AP_cluster.fit_predict(pca_wo_o)
234 print("Number of Clusters:{}".format(max(predicted)))
235 
236 from collections import Counter
237 result = Counter(predicted)
238 
239 plt.figure()
240 plt.hist(predicted,bins= max(predicted)+1)
241 plt.show()
242 display(sorted(result.items(), key=lambda x: x[0]))
243 print("CH score = {}".format(sklearn.metrics.calinski_harabaz_score(X =
         pca_wo_o ,labels = predicted )))
244 print("SH score = {}".format(silhouette_score(X = pca_wo_o ,labels =
        predicted)))
245 
246 # labelling outliers
247 from scipy.spatial import distance
248 #center can be list or array
249 #assume each center correponds to label of [0,1,2...]
250 #return a list of label
251 def classify_outliers(centers,outliers_df):
252     labels = []
253     for sample in np.array(outliers_df):
254         min_center = float('inf')
255         min_center_th = -10086
256         for i,center in enumerate(centers):
257             dst = distance.euclidean(center,sample)
258             if dst<min_center:
259                 min_center = dst
260                 min_center_th = i
261         labels.append(min_center_th)
262     return labels
263 
264 out_labels = classify_outliers(AP_cluster.cluster_centers_,pca_o)
265 sklearn.metrics.calinski_harabaz_score(X =pd.concat([pca_wo_o,pca_o])
        ,labels = list(predicted)+out_labels )
```

B.8 Model: DBSCAN

```
1
2  from sklearn.cluster import DBSCAN
3
4  # use data after pca
5  for eps in np.arange(1,25,0.5):
6      MS_cluster = DBSCAN(eps=eps,metric ="l1",min_samples = 1)
7      predicted =MS_cluster.fit_predict(data_wo_o)
8      print("Number of Clusters:{}".format(max(predicted)))
9      from collections import Counter
10     result = Counter(predicted)
11     plt.figure
12     plt.hist(predicted)
13     plt.show()
14
15 def find_optimal_eps(start,end,interval,min_sample,metric):
16     num_of_cluster = []
17     ch_score = []
18     s_score = []
19     x_axis = []
20     for i in np.arange(start,end,interval):
21
22         DBSCAN_cluster = DBSCAN(eps=i,metric = metric,min_samples =
       min_sample)
23         predicted = DBSCAN_cluster.fit_predict(pca_wo_o )
24         try:
25             ch_score.append(sklearn.metrics.calinski_harabaz_score(X =
       pca_wo_o ,labels = predicted ))
26         except:
27             continue
28         s_score.append(10*silhouette_score(X = pca_wo_o ,labels =
       predicted))
29         x_axis.append(i)
30         num_of_cluster.append(len(set(predicted) ))
31
32
33     DBSCAN_cluster_eps=plt.figure()
34
35     plt.plot(x_axis,num_of_cluster)
36     plt.plot(x_axis,ch_score)
37     plt.plot(x_axis,s_score)
38     plt.legend(["Number of clusters","CH_score","Silhouette_score"],
       bbox_to_anchor=(1.1,1))
39     plt.title("Number of clusters,Silhouette_score and CH_score with
       different damping")
40     plt.grid()
41     plt.title("Number of clusters and CH_score with different eps")
42     plt.xlabel('eps')
43     plt.ylabel('Number of Clusters/CH_score')
44     _ = plt.show()
45
46     print("Optimal eps:")
47     print(np.arange(start,end,interval)[np.argmax(ch_score)])
48     print("Optimal number for clusters:")
49     print(num_of_cluster[np.argmax(ch_score)])
50     print("Largest CH_score:")
51     print(max(ch_score))
52
53     return [ch_score,num_of_cluster]
54
```

```
for metric in ['cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan']:
    print("")
    print("Distance Metric = {}".format(metric))
    for min_sample in np.arange(1,15,1):
        print("")
        print("min_sample = {}".format(min_sample))
        try:
            ch_score_DBSCAN,num_of_cluster_DBSCAN = find_optimal_eps(1,8,0.01,min_sample,metric)
        except:
            break

def find_optimal_min_samples(start,end,interval):
    num_of_cluster = []
    ch_score = []
    s_score = []
    for i in np.arange(start,end,interval):
        DBSCAN_cluster = DBSCAN(eps = 4,min_samples=i,metric = 'manhattan')
        predicted = DBSCAN_cluster.fit_predict(pca_wo_o )
        num_of_cluster.append(len(set(predicted) ))
        ch_score.append(sklearn.metrics.calinski_harabaz_score(X = pca_wo_o ,labels = predicted ))
        s_score.append(10*silhouette_score(X = pca_wo_o ,labels = predicted))
    DBSCAN_cluster_min_samples=plt.figure()
    plt.plot(np.arange(start,end,interval),num_of_cluster)
    plt.plot(np.arange(start,end,interval),ch_score)
    plt.plot(np.arange(start,end,interval),s_score)
    plt.legend(["Number of clusters","CH_score","Silhouette_score"],bbox_to_anchor=(1.1,1))
    plt.grid()
    plt.title("Number of clusters, Silhouette_score and CH_score with different min_samples")
    plt.xlabel('min_samples')
    plt.ylabel('Number of Clusters/CH_score')
    _ = plt.show()

    print("Optimal min_samples:")
    print(np.arange(start,end,interval)[np.argmax(ch_score)])
    print("Optimal number for clusters:")
    print(num_of_cluster[np.argmax(ch_score)])
    print("Largest CH_score:")
    print(max(ch_score))

    return [ch_score,num_of_cluster]

_ = find_optimal_min_samples(2,13,1)

def find_optimal_grid():
    num_of_cluster = []
    ch_score = []
    for i in np.arange(1,300,1):
        cluster_temp = []
        ch_score_temp = []
        for j in np.arange(1,100,0.1):
            DBSCAN_cluster = DBSCAN(eps = j,min_samples=i,metric = "cosine")
            try:
                predicted = DBSCAN_cluster.fit_predict(pca_wo_o )
                cluster_temp.append(max(predicted)+1 )
```

```
                ch_score_temp.append(sklearn.metrics.
    calinski_harabaz_score(X = pca_wo_o ,labels = predicted ))
            except:
                cluster_temp.append(0)
                ch_score_temp.append(0)


        num_of_cluster.append(cluster_temp)
        ch_score.append(ch_score_temp)

    ch_score = np.array(ch_score)
    num_of_cluster = np.array(num_of_cluster)

    k = np.argmax(ch_score)
    count = 0

    for i in np.arange(2,30,1):
        for j in np.arange(1,10,0.1):
            if count==k:
                eps_o = j
                min_samples_o = i
            count+=1



    print("Optimal min_samples:")
    print(min_samples_o)
    print("Optimal eps:")
    print(eps_o)

    DBSCAN_cluster = DBSCAN(eps = eps_o,min_samples=min_samples_o)
    predicted = DBSCAN_cluster.fit_predict(pca_wo_o )
    clusters_o = max(predicted)+1
    print("Optimal number for clusters:")
    print(clusters_o)
    print("Largest CH_score:")
    print(np.max(ch_score))

    return [ch_score,num_of_cluster]

_ = find_optimal_grid()

_[0]
```

Bibliography

[1] G. Dong and H. Liu, *Feature Engineering for Machine Learning and Data Analytics*. Boca Raton, FL, USA: CRC Press, Inc., 1st ed., 2018.

[2] "Stock prediction with ml: Feature engineering." http://alphascientist.com/feature_engineering.html. Accessed: 2018-07-30.

[3] G. S. Bonde, "Extracting the best features from multi-company stock data to improve stock price prediction," 2012.

[4] Wikipedia contributors, "Principal component analysis — Wikipedia, the free encyclopedia," 2019. [Online; accessed 12-August-2019].

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] A. Singh, A. Yadav, and A. Rana, "K-means with three different distance metrics," *International Journal of Computer Applications*, vol. 67, no. 10, 2013.

[7] M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs, "Nbclust package: finding the relevant number of clusters in a dataset," *UseR! 2012*, 2012.

[8] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, vol. 344. John Wiley & Sons, 2009.

[9] M. S. Babu, N. Geethanjali, and B. Satyanarayana, "Clustering approach to stock market prediction," *International Journal of Advanced Networking and Applications*, vol. 3, no. 4, p. 1281, 2012.

[10] T. Wittman, "Time-series clustering and association analysis of financial data," *University of Texas, Austin*, 2002.

[11] M. Gavrilov, D. Anguelov, P. Indyk, and R. Motwani, "Mining the stock market: Which measure is best," in *Proc. of the 6th ACM SIGKDD*, pp. 487–496, 2000.

[12] F. Cai, N.-A. Le-Khac, and T. Kechadi, "Clustering approaches for financial data analysis: a survey," *arXiv preprint arXiv:1609.08520*, 2016.

[13] M. Momeni, M. Mohseni, and M. Soofi, "Clustering stock market companies via k-means algorithm," *Kuwait Chapter of Arabian Journal of Business and Management Review*, vol. 33, no. 2578, pp. 1–10, 2015.

[14] ttnphns (https://stats.stackexchange.com/users/3277/ttnphns), "What is an acceptable value of the calinski harabasz (ch) criterion?." Cross Validated. URL:https://stats.stackexchange.com/q/52863 (version: 2019-07-15).

[15] K. Sasirekha and P. Baby, "Agglomerative hierarchical clustering algorithm-a," *International Journal of Scientific and Research Publications*, vol. 83, p. 83, 2013.